



**IDENTIFICATION OF PREFERRED OPERATIONAL PLAN
FORCE MIXES USING A MULTIOBJECTIVE METHODOLOGY
TO OPTIMIZE RESOURCE SUITABILITY AND LIFT COST**

THESIS

David J. Wakefield, Jr., Captain, USAF

AFIT/GLM/ENS/01M-24

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

20010619 033

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense or the U.S. Government.

AFIT/GLM/ENS/01M-24

IDENTIFICATION OF PREFERRED OPERATIONAL PLAN
FORCE MIXES USING A MULTIOBJECTIVE METHODOLOGY
TO OPTIMIZE RESOURCE SUITABILITY AND LIFT COST

THESIS

Presented to the Faculty

Department of Operational Sciences

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the
Degree of Master of Science in Logistics Management

David J. Wakefield, Jr., B.S.

Captain, USAF

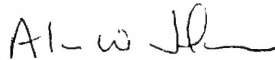
March 2001

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

IDENTIFICATION OF PREFERRED OPERATIONAL PLAN
FORCE MIXES USING A MULTIOBJECTIVE METHODOLOGY
TO OPTIMIZE RESOURCE SUITABILITY AND LIFT COST

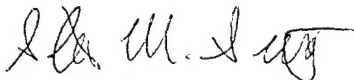
David J. Wakefield, Jr., B.S.
Captain, USAF

Approved:



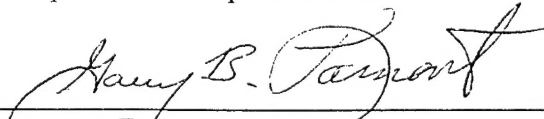
Lt Col Alan W. Johnson, Advisor
Assistant Professor of Logistics Management
Department of Operational Sciences

9 MAR 01
Date



Major Stephen M. Swartz, Co-Advisor
Assistant Professor of Logistics Management
Department of Operational Sciences

9 MAR 01
Date



Dr. Gary B. Lamont, Reader
Professor of Electrical Engineering
Department of Electrical and Computer Engineering

9 MAR 01
Date

Acknowledgements

While I am the author of this thesis, many other people have made significant contributions to its very existence.

First and foremost, I dedicate this to my wife, daughter, and son for knowing that my job is not an easy one, and doing everything they can to make it easier. Thanks also to dad, “J”, and “G” for the moral support and knowing that I’m not anti-social, I’m just an egghead. Special thanks to my mom, who can always be depended on for help.

I’m also very grateful to my thesis committee for their help during this research:

- Lt Col Alan Johnson, for keeping the leash long, letting me work through the hard stuff, and discover what it means to do research.
- Maj Stephen Swartz, for helping get into AFIT (something he may now regret).
- Dr. Gary Lamont, for inviting me into the EA fold and keeping me involved.

I was very fortunate to have a large group of supporters during my year and a half here at AFIT. I mention a few of them here:

- The other “Golden Boys”, for sticking it out with me.
- The “Back Row Gang”, for never letting me forget how late I am.
- Jonathan, for the shoes.
- Steve Oliver, Hakan Bal, and Mike Colvard, for helping me make the 2000 lab a 24 hour operation.

Finally, I am deeply indebted to Capt Jesse Zydallis, who gives of himself freely even when he himself bears a heavy burden.

David J. Wakefield, Jr.

Table of Contents

	Page
Acknowledgements.....	iv
List of Figures.....	vii
List of Tables.....	ix
Abstract.....	x
 I. Introduction.....	 1
Background	1
Problem Statement	4
Research Questions	4
Research Methodology.....	5
Assumptions	5
Scope/Limitations.....	6
Summary	7
 II. Literature Review.....	 9
Introduction	9
MOP Overview.....	9
Global Optimization.....	11
Techniques for Solving MOPs	12
Modern Methods for Handling MOPs.....	13
Local Search in Objective Space.....	14
Population Based Approach	16
Genetic Algorithms	17
Basic Operation.....	18
Initialization.....	19
Representation.....	19
Fitness Evaluation.....	21
Genetic Operators.....	21
Parameter Settings.....	25
The Schema Theorem.....	28
Constraint Handling.....	30
Multiobjective GAs.....	31
Summary	41
 III. Methodology.....	 42
Introduction	42
Model Formulation.....	42
MOP Formulation.....	45

Target MOP	49
Motivation and Objectives	53
Performance Measures	59
Experimental Design	60
Computational Environment.	61
MOMGA-II Parameter Settings	64
Summary	64
IV. Results	65
Introduction	65
Statistical Analysis	65
Absolute Performance Comparison.....	65
Execution Timing Analysis	67
Demonstration of Level-wise Nondominated Force Mix Sets.....	68
Summary	70
V. Conclusion.....	71
Introduction	71
Conclusions	73
Limitations.....	74
Recommendations	76
Future Research	77
Summary	79
Appendix A: Pareto Concepts.....	80
Appendix B: Advanced Logistics Program (ALP) Pilot Problem.....	81
Appendix C: Source Code for ENUMERATION.C.....	86
Appendix E: Source Code for Pareto_processing.c.....	97
Appendix F: Raw Data and Experimental Statistics.....	105
Appendix G: 3D Plots of PF_{true} and PF_{known} Using Alternative Parameter Values.....	111
Appendix H: 3D Plots of PF_{known} for Resource Levels 1 –5.....	116
Bibliography.....	119
Vita.....	125

List of Figures

Figure	Page
1. ALP Operational Concept	2
2. Global and Local Maxima.....	12
3. Pseudo Code for a Simple Genetic Algorithm	19
4. Example of Single-point Crossover	23
5. Pseudo Code for a Messy Genetic Algorithm	36
6. Example of Cut-n-Splice Operation.....	37
7. MOP Processing Model	43
8. Thread of Progression	44
9. MOMGA-II String Length vs. Memory Required	55
10. Three Views of the Target MOP Tri-objective Space for Resource Level 1	58
11. Problem Size vs. Execution Time	68
12. Matching Resources to Tasks.....	83
13. Kruskal-Wallis H -Test Results for Final Generational Distance	109
14. Kruskal-Wallis H -Test Results for ONVG	110
15. Plot of PF_{true} and PF_{known} for BB size 4	111
16. Plot of PF_{true} and PF_{known} for BB size 8	112
17. Plot of PF_{true} and PF_{known} for BB size 2	112
18. Plot of PF_{true} and PF_{known} for $P_{cut} = 0$	113
19. Plot of PF_{true} and PF_{known} for $P_{cut} = 0.2$	113
20. Plot of PF_{true} and PF_{known} for $P_{splice} = 0.8$	114
21. Plot of PF_{true} and PF_{known} for $P_{splice} = 0.6$	114

22. Plot of PF_{true} and PF_{known} for initial population size = 600.....	115
23. Plot of PF_{true} and PF_{known} for initial population size = 1200.....	115
24. PF_{known} Plotted With PF_{true} for Resource Level 1	116
25. PF_{known} Plotted for Resource Level 2	117
26. PF_{known} Plotted for Resource Level 3	117
27. PF_{known} Plotted for Resource Level 4	118
28. PF_{known} Plotted for Resource Level 5	118

List of Tables

Table	Page
1. Tasks	50
2. Resource Levels	50
3. Desired Task Capability Ratios	51
4. Desired Capability Matrix.....	51
5. Task Suitability / Lift Consumption Matrix.....	51
6. Experimental Parameter Settings	63
7. MOMGA-II Parameter Settings.....	63
8. Asset-Mission Task Preference Matrix.....	81
9. Raw Data for Final Generational Distance	105
10. Descriptive Statistics for Final Generational Distance	106
11. Raw Data for Overall Nondominated Vector Generation.....	107
12. Descriptive Statistics for Overall Nondominated Vector Generation.....	108

Abstract

AFIT research in support of the Advanced Logistics Project is directed at developing a Mission-Resource Value Assessment Tool for rationally assigning relative value to resources and identifying alternative force mixes to logistics and operational planners. Research of factors that affect force mix composition has been strictly limited to how the operating environment of USAF combat aircraft influences their performance in specified aerospace missions. In contrast, this research makes use of an aircraft's designed suitability to perform specified aerospace missions in order to examine the tradeoff between mission suitability and the amount of lift needed to deploy and operate the asset.

An Evolutionary approach was applied to a tri-objective constrained optimization problem with 15 decision variables with the goal of producing five Pareto optimal sets of force mixes corresponding to five progressively larger sortie capability levels. Analysis of the results includes absolute performance comparisons using different operating parameter settings, and time complexity in relation to problem scale. Preliminary results were also generated from a version of the algorithm that uses a solution repair function. These results help to assess the viability of using a multi-objective fast messy genetic algorithm to identify well balanced force mixes.

Know your limits. Now shatter them.

djw

IDENTIFICATION OF PREFERRED OPERATIONAL PLAN
FORCE MIXES USING A MULTIOBJECTIVE METHODOLOGY
TO OPTIMIZE RESOURCE SUITABILITY AND LIFT COST

I. Introduction

Background

Death by inventory is the concept of stockpiling excessive inventory to compensate for poor logistics management. This is a difficult and expensive business practice, and unfortunately, is employed in DoD operations. Take, for example, the strategic movement of personnel and equipment during the Gulf War. Although branded a success by our leaders in DoD, the war served to highlight logistical problems. “One of USTRANSCOM’s most intractable and high-visibility problems during Desert Shield/Desert Storm was a backlog of sustainment cargo at aerial ports of embarkation, primarily in the United States” (Matthews and Holt, 1996:84). The amount of arriving cargo also overwhelmed the destination points. “Half of the 40,000 bulk containers shipped into the theater had to be opened in order to identify their contents, and most of it failed to contribute in any way to our success on the battlefield” (Muczyk, 1997:89). These problems illustrate a serious gap in what the combatant commander or operator wants to accomplish and what the logistician can make available. This disconnect between operations and logistics is important, as Paul Judge notes:

The warfighting commander demands visibility of assets and requires confidence in rapid availability. Without direct knowledge that commodities and reparables

The deliverable program is expected to give logistics planners a tool that uses real-time data to rapidly develop a campaign specific logistics plan and perform dynamic replanning (Carrico, 2000).

ALP leverages the revolution in technology to narrow the gap between operations and logistics, giving planners the capability to review multiple deployment plans that are based on current information. But when comparing numerous plans and scenarios, how does one answer, “What is best?” In other words, from a pool of available resources, is there a mix of those resources that would be preferred by the combatant commander over all others? AFIT research in support of ALP is directed at developing a Mission-Resource Value Assessment Tool (M-R VAT) that “rationally assigns relative value to material resources” (Swartz, 1999) and identifies alternative force mix compositions to planners.

A *Mission Ready Resource* (MRR) is a combination of an asset type and its resources, e.g. aircraft, pilot, fuel, munitions, support equipment and personnel, etc., that is designed to have a certain *suitability* for a single task. A combination of MRR types is defined to be a *MRR set* or *force mix*. To demonstrate, assume that a notional aircraft F , has two configurations, F_A and F_B , which constitutes two MRR types. Further, if the aircraft could be prepared and flown three times per day, then it would represent three MRRs per day. These three MRR’s could either be all F_A configurations, or all F_B configurations, or some combination of the two configurations. A MRR is *consumed* in the performance of its task, i.e. fuel, munitions, engine cycles, etc. The goal of a logistics plan is to provide the combatant commander with the Mission Ready Resource (MRR) sets that satisfy the time-phased need for those resources. In other words, the

combatant commander desires to fulfill a task with the best combination of MRRs that is dependent on resource suitability to a given task, resource level, and time, along with theater-specific factors. There is most likely a number of acceptable MRR sets for which certain combinations of MRRs may be assessed as having equivalent suitability. It is desirable for the planner to select the MRR combination that best maximizes the time- and resource level dependent suitability.

An MRR set provides a certain suitability and capability to the combatant commander, but at a cost: consumption of lift resources. The finite lift capability of the U.S. military is a key constraint on the amount and timing of resources flowing into a target area. Therefore, it is desirable to deliver an MRR combination that minimizes lift. There are then two competing objectives that the planner must deal with in order to present the best MRR combination to the decision maker: maximize asset suitability and minimize lift consumption.

Problem Statement

Given a choice among time-phased asset sets, simultaneously minimize lift resource consumption (cost) and maximize asset set suitability over time.

Research Questions

1. Which methodologies can be used to trade-off lift cost and asset suitability and result in an asset mix that is preferred over others?
2. What are the forms of decision and objective spaces?
3. How should the multiobjective optimization approach be evaluated?

4. Does the selected approach result in an acceptable solution to the research problem in a reasonable amount of time?

Research Methodology

The two objectives, maximize suitability/minimize lift, are in conflict, so it may be that no single optimum solution exists with respect to both objectives. It is desirable to ascertain whether an exact solution can be obtained within reasonable time. If not, an acceptable compromise solution must be found. Candidate approaches exist and are well documented in literature. Candidate approach selection is based on its applicability to the problem, its overall utility, and its utilization of computational resources.

For any acceptable asset set, the combination of lift costs for all asset sets over a given period is constrained by the maximum throughput of the transportation pipeline during that period. For the initial model formulation, it is assumed that the combined lift will not exceed the maximum available lift. A constraint can be added to subsequent models in a way that limits asset set selection to allocated lift consumption.

Assumptions

The combinatorial nature of this research necessitates the use of a relatively small set of assets with which to explore the algorithmic search for an acceptable solution. Additionally, actual asset capability with regard to specific missions may be classified. Therefore, this research is constructed around a notional Air Expeditionary Force, comparable in size and diversity to that depicted in the ALP Pilot Problem (Swartz, 1999). No actual assessments of aircraft suitability or lift cost will be used. The results

of this research may be sensitive to scale in terms of time required to produce a set of acceptable solutions.

It is difficult to determine what the actual logistical footprint is for a given asset set. The logistics support for a force is often undefined until just hours prior to movement (Judge, 1998:32). It was noted during Desert Storm that “the actual material shipped grew in size without anyone’s knowledge and certainly without any tools to predict the eventual impact” (Lynn, 1997:15). To be of use, this research assumes that asset set lift consumption is both sufficiently accurate and available for planning. Research conducted by Matt Goddard suggests that, for F-16s, the relationship of asset quantity to consumption is linear (2001).

Scope/Limitations

In their research on a campaign planning decision support tool, Christopher Buzo and Paul Filcek proposed a comparison of competing sets of combat aircraft assets based on two criteria (Buzo, 2000:2, Filcek, 2001). The first criterion is the *intrinsic suitability* of a MRR to one or more specific missions. For instance, a properly configured F-16C is capable of adequately performing many roles such as Air Interception, Short-Range Reconnaissance, and Air-to-Air. In contrast, a B-2’s intrinsic suitability is relatively confined to Strategic Bombing.

The second criterion is based upon situation specific or *extrinsic* factors of the campaign itself. Such factors modify the task suitability of MRRs in certain scenarios. For example, a planner with knowledge of an enemy with a high anti-air capability would alter a force mix in favor of aircraft with a high absolute suitability in air defense

suppression. Political issues are also extrinsic factors of a campaign, forcing planners to consider over-flight rules, coalition participation, and beddown constraints, to name a few.

Buzo's and Filcek's research of factors that would affect force mixes was strictly limited to extrinsic factors of USAF combat aircraft to perform specific aerospace missions (2000, 2001). In contrast, this research makes use of the intrinsic suitability of MRRs to examine the tradeoff of between MRR suitability against MRR lift cost—extrinsic factors are not considered.

Discussion of operational plans and actual task suitability may be classified. Therefore, this research and its conclusion is restricted to the unclassified realm.

Summary

This chapter illustrated the disconnect between operations planners and logistics planners and ALP's intention to close that gap. AFIT research is centered on providing a decision support tool that would help campaign planners select the best force mix from a pool of combinations. The problem of force mix selection can be handled as a multiobjective optimization of two competing objectives: minimize lift resource consumption (cost) and maximize MRR task suitability. This research proposes to develop a methodology for presenting the decision maker with a set of acceptable force mixes after which an extrinsic assessment is then made and the user-defined best force mix is selected.

Chapter II provides a background on multiobjective optimization, discusses global versus local optimization, and reviews classic and modern multiobjective techniques.

Chapter III describes the methodology used to construct the multiobjective problem and evaluate the solution approach. Chapter IV details the results using the selected multiobjective optimization methodology. Chapter V provides conclusions on research contributions and makes recommendations for further research.

II. Literature Review

Introduction

Mathematical optimization is the formal title given to the branch of computational science that seeks to answer the question ‘What is best?’ for problems in which the quality of any answer can be expressed as a numerical value. Such problems arise in all areas of mathematics, the physical, chemical, and biological sciences, engineering, architecture, economics, and management, and the range of techniques available to solve them is nearly as wide. In practical problems, we often want to optimize more than one measure of performance at once.

This research is concerned with a particular problem class: multiobjective optimization problems (MOPs). The purpose of this chapter is to provide a broad overview of the field in order answer the first research question: “which methodologies can be used to trade-off lift cost and asset suitability and result in an asset mix that is preferred over others?” Following an overview on the MOP class, modern approaches to handling MOPs are presented.

MOP Overview

The goal of an optimization problem can be formulated as follows: find the combination of parameters (independent variables) that optimize (maximize or minimize) a given quantity, possibly subject to some restrictions on the allowed parameter ranges. The *objective function* is the quantity to be optimized. The parameters that can be changed are the *decision variables* that represent discrete solutions of a combinatorial

problem (Reeves, 1995:2). The restrictions on parameter values are termed *constraints*. For convenience of mathematical treatment, all problems in this thesis are assumed to have minimization objectives unless stated otherwise.

To solve a decision-making problem analytically, it is helpful to state the problem in numerical terms. Given that an objective, O , has a corresponding n -dimensional set of alternatives, X , the criterion for the objective is defined as an objective function:

$$f : X \rightarrow \mathbb{R}^1 \quad (2.1)$$

where f is a mapping that may be linear or non-linear (Van Veldhuizen, 1999:2-2). The general form of an MOP with p objectives is

minimize:

$$f(x) = (f_1(x), f_2(x), \dots, f_p(x)) \text{ over } x \in X \quad (2.2)$$

subject to:

$$c_i(x) = 0 \quad i = 1, 2, \dots, m' \quad (2.3)$$

$$c_i(x) \geq 0 \quad i = m' + 1, \dots, m \quad (2.4)$$

where x is the column vector of the n independent variables, and $c_i(x)$ is the set of constraint functions that, depending on the situation, may or may not be included in the problem (Sawaragi, et al., 1985:2).

In an MOP, it is difficult to obtain a unique optimal solution. This is because the problem's objectives are usually in conflict with one another: one cannot improve the performance of a particular objective without causing a corresponding deterioration in performance in one or all of the others. Examples of conflicting objectives may be maximizing speed and safety in a vehicle, or minimizing acquisition cost and schedule of

a new aircraft while maximizing its performance. Classical methods of dealing with this problem produce a solution by combining objectives in some way that is usually a subjective expression of an *a priori* not well understood trade-off surface (Fonseca and Fleming, 1993:1, Horn et al. 1993:1, Deb, 1999b:4, Ehrgott and Gandibleux, 2000:12, Taber et al., 1999:1). By treating the problem as multiple, competing objectives, the result is a set of solutions in decision variable space whose components represent a trade-off in the objective function space. A *decision maker* implicitly chooses an acceptable solution (or solutions) by selecting one or more of these alternatives (Van Veldhuizen, 1999:2-2). Ideally, the alternatives should be selected from a set of equally preferred solutions called the *Pareto optimal set*, or simply *P*. Pareto optimal solutions are also termed *efficient* or *admissible* solutions (Yu, 1985:22). The mapping of *P* to the objective space forms the *Pareto front*, *PF*. The Pareto front is also known as the *nondominated* front. The reader is referred to Appendix A for additional discussion of Pareto concepts.

Global Optimization. The desired solution to an optimization problem is the true or *global optimum*. The strict definition of the global optimum (minimum) x' of $f(x)$ is

$$f(x') < f(y) \text{ for all } y \in V(x), y \neq x', \quad (2.5)$$

where $V(x)$ is the set of feasible values of the decision variables x (Allen, et al., 1996). A complication that arises in nonlinear optimization is that a local optimum need not be a global optimum. For example, consider the function of a single variable plotted in Figure 2. Over the interval $0 \leq x \leq 20$, this function has three local maxima— $x = 3.8$, $x = 10.2$, and $x = 16.7$ —but only one of these— $x = 3.8$ —is the global maximum.

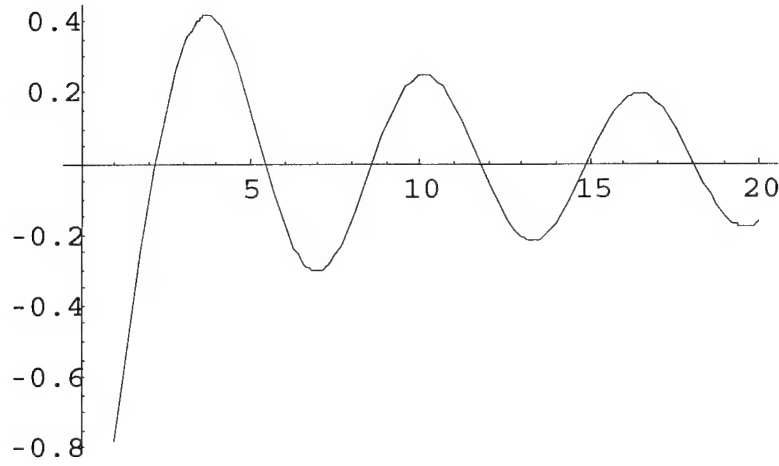


Figure 2. Global and Local Maxima

Finding the global optimum of a general mixed integer MOP is *NP-Complete* and MOP solutions that satisfy all constraints and globally optimize all objective functions may not even exist (Van Veldhuizen, 1999:2-2).

Techniques for Solving MOPs. Techniques for solving MOPs have existed for years and usually rely upon either enumerative or approximation approaches (Ehrgott and Gandibleux, 2000:12, 20). For MOPs, it is from the Pareto optimal set that an informed decision maker chooses a compromise solution. Ideally, the presented set is the true Pareto optimal set P_{true} that corresponds with the true Pareto front PF_{true} . However, complete enumeration of solutions for even a reasonable MOP is impractical from both computational and decision making standpoints.

A MOP is required to pare down the solution set to one in which the decision maker can feasibly use to select a solution that represents the best tradeoff between objectives. In a real-world problem with real-valued solutions, the presented Pareto optimal set P_{known} is a discretized approximation or *subset* of a continuous P_{true} .

According to Reeves, this kind of solution tends to favor an approximate or heuristic approach to finding it (1995:11). While an exact model to a real-world problem is beyond our reach, “it may be possible to model the real-world problem rather more accurately than is possible than if an exact algorithm is used” (Reeves, 1995:11). A heuristic allows us to solve optimization problems “in ways that are less than perfect yet of considerable practical value” (Harel, 1987:344).

Deterministic heuristics, whose members include, but are not limited to, greedy algorithms, descent algorithms, and deterministic linear / non-linear programming methods, use problem domain knowledge to shrink the solution space (Van Veldhuizen, 1999:2-10). For any heuristic, a reduction in computational cost comes without being able to guarantee either feasibility or optimality. Further, deterministic algorithms, when applied to MOPs, suffer from their poor handling of irregularities in the search space—high-dimensional, discontinuous, multimodal, and / or *NP*-Complete—and can be expected to produce only local solutions (Van Veldhuizen, 1999:1-3, 2-12, Reeves, 1995:6). MOPs are usually better handled by flexible and more robust heuristic approaches (Reeves: 1995:11).

When systematic search methods fail, stochastic techniques are used. The algorithms discussed in the following sections all employ some form of random search.

Modern Methods for Handling MOPs

Heuristic approaches are typically designed for a specific problem and are not suited for a wide range of applications (Ehrgott and Gandibleux, 2000:15). In contrast, a metaheuristic employs a master strategy to take advantage of the search space and guide

the search (Ehr Gott and Gandibleux, 2000:15). Metaheuristics are much more general in their application. Recent advances in computational power have pushed metaheuristics to the forefront. Simulated Annealing (SA), Tabu Search (TS), and Evolutionary Algorithms (EAs) are examples of metaheuristics that have been researched in great depth and have produced superior results, in terms of both solution quality and computational efficiency, in a wide variety of applications (Ehr Gott and Gandibleux, 2000:15). All methods rely heavily on computing skills for practical implementation.

There are two main approaches used by metaheuristics: 1) local search in objective space and 2) population based.

Local Search in Objective Space. Based on the principle of search directions, this approach starts from some initial solution and proceeds in a given search direction to focus on a portion of the nondominated front. The search proceeds iteratively in other search directions in order to approximate the entire Pareto front. “At any time the search mechanism uses only one solution and an iteration tries to attract the solution generated” towards the Pareto front along the given direction (Ehr Gott and Gandibleux, 2000:16).

Hill Climbing, Simulated Annealing (SA) and Tabu Search are examples of the first approach. Hill climbing begins with a single random solution that is perturbed to change its evaluation. After several such perturbations, the best evaluation is chosen as the next starting point. Continuing on in this way eventually results in reaching an optimum. However, it is not known whether it has reached local or global optimum. The search space can be explored by starting repeatedly with a new random solution in hopes of finding a better solution (Caryl, no date).

SA employs an analogy between the way in which a metal cools and freezes into a minimum energy crystalline structure, and the search for a minimum in a general system. At high temperatures, the molecules of a liquid move freely. As the liquid cools, that mobility becomes restricted and the molecules achieve crystalline form and the system's minimum energy state, i.e. global optimum.

Like hill climbing, SA randomly perturbs the objective function in a way that will, for a given change, causes a decrease and for another change causes an increase. But instead of selecting the best evaluation and continuing on (analogous to rapidly cooling the liquid), SA introduces an element of randomness: a change that does not improve upon the current optimum is executed with a probability $p < 1$. This is typically based on the Boltzmann probability distribution:

$$P(E) \sim e^{-\frac{E}{kT}}$$

where E is the energy of the state, k is Boltzmann's constant, and T is the temperature. This equation means that the probability of finding a particle with energy E is proportional to the exponential of $-E$ divided by the product of k and T (Rappe, no date). So at a given temperature, a system can be in a range of possible energy states. A higher temperature increases the likelihood of a high energy state. Simulated annealing makes use of the fact that at low temperatures, there is still of chance of being in a high energy state, thus allowing a *jump* out of a local minima. This random perturbation gives an SA its ability to avoid being trapped at a local optimum (Reeves, 1995:26).

An SA implementation tends to be problem specific. The choice of the temperature or *annealing* schedule depends on the expected range of function values and

the shape of the function surface. Experimentation is required to obtain a method that works well for a particular problem.

TS is can also be employed as a stochastic method, but rather than using random moves, TS employs a directed search along with a memory to imitate intelligent processes (Reeves, 1995:13). TS is a form of neighborhood search. Beginning with a solution within a defined neighborhood, the algorithm proceeds iteratively to visit a series of locally optimal solutions. At each iteration, a best neighbor is chosen to replace the current solution. To allow the search to move beyond local optima, a list of moves that are not allowed or *tabu* is used. This list prevents recently visited solutions from being considered for a given number of iterations of the algorithm (Reeves, 1995:83, 86-88).

Population Based Approach. The population based approach takes advantage of information carried by a population of solutions. Heuristics using this method predominantly fall under the category of *Evolutionary Algorithms* (EAs), a class that uses the evolutionary concepts of *survival of the fittest* and *generational improvement* as its inspiration.

Despite the probabilistic nature of EA operators, EAs are not completely random searches and are directed by the information carried by the population. Contrary to the local search methods, where only one individual is attracted toward the Pareto front, here the entire population contributes to the evolutionary process toward the Pareto front by searching for many nondominated solutions at once. It is this characteristic that makes the population based approach very attractive for solving MOPs, but it comes at a relatively higher computational cost since many fitness evaluations are required (Ehrgott and Gandibleux, 2000:16, Fonseca and Fleming, 1993:1).

Recent research by Van Veldhuizen catalogued 206 multiobjective EAs (MOEAs) (1999:A-1). A widespread implementation of the population-based approach is a type of EA called the Genetic Algorithm (GA). Its broad applicability, ease of use, and success in handling MOPs makes it “no surprise that a number of different multi-objective GA implementations exist in the literature...” (Deb, 1998:4). A GA, in a single run, can provide “a number of Pareto-optimal solutions” (Deb, 1998:26) and has the “ability to find global optima while being able to cope with discontinuous and noisy functions” (Fonseca and Fleming, 1993:7). A detailed discussion of GA theory, operation, and types of MOEAs can be found in the next section.

The limit of any optimization method is succinctly expressed by the *No Free Lunch* theorems (Wolpert and Macready, 1996), which tell us that “used blindly, there is an equal chance that any optimization technique will perform the same” (Practical Guide, no date). The choice of which optimization method to use should be based on what is known about the system being optimized. Even though a GA has no guarantee of performing better than another method in a given application, in most cases a GA’s parameters and configuration can be tailored to achieve adequate search performance.

Genetic Algorithms

The Genetic Algorithm (GA) is based on the biological processes of evolution as described by Charles Darwin (1936). An organism is made up of genetic material embedded with environmental knowledge. Natural selection sees to it that those individuals that are better suited to their environment survive to pass-on their *good* genetic material. The less fortunate die and take their bad genes with them.

Reproduction happens in an environment where the selection of who gets to mate is largely a function of individual fitness. Reproducing pairs or *parents*, produce offspring with chromosomes containing information from each parent. Evolution uses mutation to stimulate diversity in the population. Mutated individuals do not always survive, but occasionally there are those that are better suited to their environment and more competitive than the others. Their environmental advantage is passed on to their offspring and ultimately to future generations (Caryl, no date).

Basic Operation. There are five characteristic components in every GA (Caryl):

1. A way to create an initial population of potential solutions
2. A genetic representation for solutions to the problem
3. An evaluation function that plays the role of the environment, rating solutions in terms of their fitness
4. Genetic operators that alter the composition of children during reproduction
5. Values for various parameters that the genetic algorithm uses (population size, probability of applying genetic operators)

The pseudo code for the basic algorithm is presented in Figure 3:

```
Start GA
    // start with an initial time
     $t = 0$ ;
    // initialize random population of individuals
    initialize  $P(t)$ ;
    // evaluate fitness of all initial individuals of population
    evaluate  $P(t)$ ;
    // test for termination criterion
    while not done do
        // increment the time counter
         $t = t + 1$ ;
        // select a sub-population for offspring production
         $P'(t) = \text{parents from } P(t)$ ;
```

```

        // recombine the "genes" of selected parents
        recombine  $P'(t)$ ;
        // perturb the mated population stochastically
        mutate  $P'(t)$ ;
        // evaluate its new fitness
        evaluate  $P'(t)$ ;
        // select the survivors based on fitness
         $P(t) = \text{survive } P(t), P'(t)$ ;
    end do
end GA.

```

Figure 3. Pseudo Code for a Simple Genetic Algorithm (Osyczka and Kundu, 1995:95)

A single iteration of the *while* loop constitutes a *generation*.

Initialization. A GA maintains a population of solutions, all of which are potential parents. The first generation's population is initialized, usually with randomly generated individuals. Another technique is to initialize the population with high-quality solutions. This approach has shown to increase the speed of convergence, but with an increased possibility of premature convergence to only a portion of the Pareto front (Reeves, 1995:164).

Representation. A GA can be thought of as a "DNA simulator" where nonbiological structures could be modeled in terms of bit strings that could be changed through transformations analogous to evolution. In the traditional GA, a genotypic representation scheme encodes n decision variables into n sequences of binary bits that together form a bit string or chromosome that represents an individual solution in decision space. The values of the variables are termed alleles and the location of a bit within the string is its locus (Reeves, 1995:138).

Genotype is the genetic phrase for the encoded variables while phenotype is used for decoded variable representation. This encoding scheme leads naturally to

representation of integer decision variables. For example, equation (2.6) is an objective function in two variables:

$$\max: f(x, y) = x^2 + y^2 \quad (2.6)$$

Each variable can be represented by bit string of length m . If the phenotypic representations for x and y are 3 and 6 respectively (i.e. $x = 3$ and $y = 6$ and $m = 4$, then (2.7) is the 8-bit string genotypic representation of x and y :

$$0\ 0\ 1\ 1\ 0\ 1\ 1\ 0 \quad (2.7)$$

As this also represents an entire solution to equation (2.6), the bit string is itself a chromosome.

Continuous variables are approximated through a scaling function. The accuracy with which an optimum solution can be resolved depends on the length of the bit string. For instance, a variable represented by a 22 bit chromosome can range between 0 and $2^{22} - 1$. When rescaled into real numbers with a range of 3, this representation gives quantization errors of 7.2×10^{-7} . The scaling operation also serves to designate the upper and lower bounds of the decision variable (Osyczka and Kundu, 1995: 95).

Other encoding methods that have been used are real number representation and an integer representation variation. In real number representation, the each decision variable is simply represented by floating point number, resulting in a “one gene / one variable relationship” (Practical Guide, no date). Using an integer instead of a floating-point number results in an integer representation. The disadvantage in using a binary representation is a result of the extra steps needed to decode the binary string to a floating-point number and back for each fitness evaluation (Practical Guide, no date). However, contrary to real number representation, the theoretic aspects of binary

representation have been thoroughly explored, placing performance evaluation of the GA on much more solid ground (Practical Guide, no date).

Fitness Evaluation. Since the goal of optimization is to either maximize or minimize the objective function value, a measure of solution fitness is a function of the objective function value. If the objective function in equation (2.6) is used as a direct measure of solution fitness, then the chromosome in equation (2.7) can be decomposed into x and y values and substituted into the objective function:

$$f(3,6) = 3^2 + 6^2 = 45$$

If maximizing, the solution (3, 6) would yield a better fitness than solutions with lesser values.

Once a population has been produced, it can be evaluated using an objective function that characterizes every individual's performance in the problem domain. The number of fitness evaluations increases with the number of objective functions and population size. Fitness evaluation is the primary source of GA computational cost.

Genetic Operators. Traditional genetic operators are selection, recombination, mutation, and evaluation. Selection for reproduction is dependent on the evaluated fitness of each solution, the idea being that better solutions have better representation in the population in order to improve the population with respect to preceding generations. Selected solutions are recombined to form new solutions that are then evaluated for inclusion in the next generation's population. Prior to evaluation, a solution may be changed by probabilistically applying a mutation operation to one or more of its genes.

Selection. Solution fitness is used to bias the selection process toward highly fit individuals while still allowing less fit individuals to reproduce. This has the

effect of keeping a measure of diversity in the population thereby making the search more global. Highly fit individuals are given a higher probability of being selected for reproduction than individuals with a lower fitness value. The average performance of individuals can be expected to increase since those individuals with better fitnesses are more likely to be selected for reproduction and the lower fitness individuals are eventually culled from the population. Individuals may be selected more than once at each iteration of the GA.

There are a variety of selection schemes employed in GAs (Osyczka and Kundu, 1995:95, Deb, 1999a:7). Common methods include proportionate selection, ranking selection, and tournament selection. In proportionate selection, the probability that an individual is chosen for selection is the individual's fitness divided by the sum of the current population's fitnesses:

$$p_{select}(i) = \frac{fitness(i)}{\sum_{j=1}^n fitness(j)} \quad (2.8)$$

where i is the individual in question and n is the population size. The new population of potential parents is then selected by making n random draws from a uniform distribution (Dorigo and Maniezzo, 1993:7).

In ranking selection, the population is sorted from best to worst in terms of fitness. The number of copies that an individual should receive is given by an assignment function, and is proportional to the rank of an individual rather than its absolute fitness value.

In tournament selection, successive groups of k of individuals are chosen from the population and compared. The best individual from each group is selected as a parent for

the next generation. This process is repeated until the mating pool is filled. When $k = 2$, each solution will compete twice. The best solutions will have two copies in the new population, the worst are eliminated, and those in between have one. Deb reports that “tournament selection has better convergence and computational time complexity properties compared to any other reproduction operator that exist in the literature, when used in isolation” (1999a:7).

Recombination. The recombination operation is typically referred to as a *crossover*. After selection, each individual has a probability p_c , called the *crossover rate*, of being chosen for crossover. This probability is usually set high, between 0.5 and 0.9. Randomly chosen pairs of individuals are combined to produce two offspring. Crossover can be applied in different ways. Two of the most often used crossover operators are single point and multi-point crossover.

In single point crossover, a common point between two genes in both parents is selected at random. The number of points is simply the length of the chromosome, l , minus one. The offspring are created by concatenating the pre-selection point genes from one parent with the post-selection point genes from the other parent. For example, if the randomly selected crossover point is between the third and fourth genes of the chromosome, as in Figure 4, parents $P1$ and $P2$ produce the offspring $O1$ and $O2$ (Reeves, 1995: 154).

$P1$	1 0 1 0 0 1 0	$O1$	1 0 1 1 0 0 1
	↑		
$P2$	0 1 1 1 0 0 1	$O2$	0 1 1 0 0 1 0
	↑		

Figure 4. Example of Single-point Crossover

In multi-point crossover, up to $n - 1$ points may be selected as crossover points, n being the population size. The parents can then swap every other segment. In uniform crossover, each gene position is considered for crossover. However, this method has a high probability of producing offspring that are considerably different from their parents and so p_c is usually set low, e.g. 0.01 (Practical Guide, no date).

In practice, “the simple crossover operator has proved extremely effective” (Reeves, 1995:170). It does no backtracking or table lookups, making it a simple and efficient method to implement.

Mutation. Crossover is responsible for the search aspect of the GA and is considered the primary operator. Mutation is responsible for keeping a measure of diversity in the search and is considered as a background operator. The offspring from reproduction are further perturbed by mutation. Each bit in a chromosome is changed with given probability p_m , called the *mutation rate*. In a binary representation scheme, this means flipping the bit.

The mutation operator is both simple and powerful by guaranteeing “that every point in the search space can be reached” (Dorigo and Maniezzo, 1993:7). It works by biasing the creation of new solution in the neighborhood of the original solution (Deb, 1999a:10). Overuse of the mutation operator would destroy this relationship and so it is used sparingly, with p_m usually set to between 0.1 and 0.001. Both the Messy GA (mGA) and Multiobjective Messy GA (MOMGA) set p_m to 0 and have obtained highly competitive results in comparison with other GA implementations (Van Veldhuizen, 1999).

At this point, the next generation's population must be filled. The process described by Osyczka and Kundu in Figure 3 is only one example of how this may be accomplished. Following along, the new individuals that result from crossover and mutation are evaluated for fitness and compared with the parent population. The next generation is made up of the individuals from both generations with the highest fitnesses. Deb implements a variation on filling the next generation by using the $n(p_c)$ offspring and $n(1 - p_c)$ parents (Deb, 1999a:9). This method is expected to produce better solutions since the higher fitness parents that are the result of the selection process are used to fill the vacancies left by the crossover operator in the new population. The $n(1 - p_c)$ parents can be copied either deterministically or at random.

These processes of selection, recombination, mutation, and evaluation are then repeated until some termination criteria is satisfied, e.g. upon reaching a maximum number of generations, a specified fitness, a specified number of solutions in the nondominated solution set, or after an elapsed period of time. Another stopping rule used is based on the number of fitness function evaluations performed. The number of function evaluations required to find the optimal solution set, within a given tolerance of course, is an important measure of algorithm efficiency (Van Veldhuizen and Lamont, 2000(a):141). This is discussed in the section on multiobjective GAs.

Parameter Settings. The parameters most often cited as having a significant affect on the performance of an EA are population size (n), crossover rate (p_c), and mutation rate (p_m) (Gray, 1997, Caryl, no date, Practical Guide, no date). Despite the many papers on the theoretical and applied use of EAs, there are very few quantitative methods for determining the proper values to use in a given optimization problem

(Practical Guide, no date). The parameter values that produce the most efficient and effective results depend upon the given problem and how the EA is applied, i.e. search space topology, representation scheme, selection and recombination methods. The parameters may even vary with each generation or between decision variables (Mathematical Optimization, no date).

Researchers have used parametric studies to determine the best settings for a particular problem (Deb and Agrawal, 1999:3, Practical Guide, no date). Ad hoc parameter settings are based on what is generally known about how their interaction affects a GA's performance, i.e., *algorithmic efficiency* and the *exploration* and *exploitation* of the search space. For GAs, the most time-consuming task is fitness evaluation (Van Veldhuizen and Lamont, 2000(a):142). GA complexity and efficiency are generally stated in terms of the number of fitness evaluations performed. Search space exploration refers to how well population diversity is maintained in the nondominated front. Search space exploitation refers to how well the search is guided towards the true Pareto front (Deb, 1998:4). The parameter settings used depend on what aspect of GA performance the researcher is focused on (Deb and Agrawal, 1999:2-3). Swinging a parameter's settings through a predetermined minimum and maximum can give some picture of Pareto front. In consideration of the interaction between parameter settings, the number of runs needed to do this is itself a multiobjective problem and is NP-complete (Zydallis, 2001).

It can be seen intuitively that the setting of the population size is a trade-off between solution diversity and algorithmic efficiency. As n increases, the diversity of among individuals is expected to increase, thereby decreasing the chance of *premature*

convergence to suboptimal solutions or only a portion of the Pareto front. This does not imply that increasing the population size automatically improves convergence to the Pareto front (Zitzler et al., 1999:18). Increasing the population size does have a computation cost. For a k objective optimization problem, at least kn fitness evaluations are required. Suggested values for n are between 25 and 100 (Practical Guide, no date).

Deb and Agrawal have shown that for simple functions, GAs using both crossover and mutation perform better than either of them alone, and suggest the use of a large crossover probability with a small mutation probability (1999:20). With more difficult problems, the use of crossover exclusively (along with a suitable population size) was shown to be effective (Deb and Agrawal, 1999:20). This is not surprising since crossover is the key search operator and implicitly manipulates the best substrings or *building blocks* to create Pareto optimal solutions (this is discussed further in the section on the Schema Theorem). This does not imply that mutation is unimportant. Mutation is used to uncover building blocks from which the crossover operator may direct the search away (premature convergence). Too large a rate may destroy the information carried by building blocks. Depending on how much pressure the researcher wants to apply to Pareto front distribution, suggested rates for mutation range between 0.001 and 0.1 (Practical Guide, no date). De Jong's work with GAs on problems with discontinuities, high dimensionality, noise, and multimodality suggests that settings of $n = 50$, $p_c = 0.60$, and $p_m = 0.001$ would give adequate results in most cases (Mathematical Optimization, no date). A commonly held opinion regarding parameter settings is that "although there is no unique combination guaranteeing *good* performance, choosing wisely may well result in more effective and efficient implementations" (Van Veldhuizen, 1999:2-18).

Quantitative methods for determining the population size have been derived in the literature but Deb and Agrawal have recognized that what is needed is a “good yet ready-to-use population sizing estimate for generic problems” (1999:21). For crossover based GAs, Goldberg, Deb, and Clark derive an estimate for the minimum population size, N_s , needed to trigger correct building block processing (Deb and Agrawal, 1999:11-12):

$$N_s = 2c\kappa \frac{\sigma_M^2}{d^2} \quad (2.9)$$

where c is the tail of the Gaussian distribution relating to the permissible error rate α , κ , is the number of competing schemata, and σ_M^2 / d^2 is the inverse of the signal-to-noise in the underlying problem.

Both De Jong and Hessner and Manner suggest quantitative methods for determining the mutation rate, suggesting that the rate is inversely proportional to the population size. The Hessner and Manner formulation is

$$p_m = \frac{1}{n\sqrt{l}} \quad (2.10)$$

where n is the population size and l is the length of the chromosome (Practical Guide, no date).

The Schema Theorem. Chromosomal representation allows the manipulation of information about the search space and the transfer of that information to other chromosomes. This information is carried in the substrings of the chromosome. Thus, each substring represents a subspace solution (Osyczka and Kundu, 1995:95). Substrings are grouped based on similarity at certain string positions, called *schema*, and are represented on a template of 0's, 1's, and *'s (in a binary representation) (Dorigo and

Maniezzo, 1993:8). The “*” is a wildcard symbol that represents both 0 and 1. Thus a schema $S_1 = (1\ 0\ *\ *)$ represents strings with a 1 in the first position and a 0 in the second position.

The *length* of a schema, $\delta(S)$, is defined as the distance between the two most distant symbols in the schema that are not wildcards. The *order* of a schema, $o(S)$, is defined as the number of wildcards subtracted from the number of symbols in the schema (Dorigo and Maniezzo, 1993:8). So $\delta(S_1)$ is $2 - 1 = 1$ and $o(S_1)$ is $4 - 2 = 2$. The *fitness* of a schema is the average fitness of all strings that match the schema (Osyczka and Kundu, 1995:95).

Since a schema is a grouping of similar strings, it represents a region in the search space. For the objective function in equation (2.6), the schema S_1 represents strings with x and y values varying from 8 to 11 with function values varying from 128 to 242. A schema $S_2 = (0\ 0\ *\ *)$ would result in function values varying from 0 to 18. Since the objective is to maximize, strings similar to S_1 are preferred and increase in proportion over those like S_2 . This is given by Holland’s Schema Theorem, which formalizes the expected number m of schemata h within a population at generation t :

$$m(h, t+1) \geq m(h, t) \frac{f(h)}{f} \left[1 - p_c \frac{\delta(h)}{l-1} - p_m o(h) \right] \quad (2.11)$$

where $f(h)$ is the average fitness of all strings similar to h within the population, f is the average fitness of the population, and the rest as defined previously (Goldberg, et al., 1989:5). It can be seen from equation (2.11) that short, low-order, above-average fitness schemata, or *building blocks*, are desirable if a schema is to grow in subsequent generations. Building blocks, according to Goldberg, “will increase in number with

exponential speed” (Dorigo and Maniezzo, 1993:9-10). That a GA accomplishes this implicitly through the selective pressure fostered by representation schemes and genetic operators is postulated by Goldberg in what is known as the *Building Block Hypothesis* (Deb, 1999a:14, Van Veldhuizen, 1999:4-3).

Constraint Handling. Most real world problems are going to be constrained in some way (time, money, space, bandwidth, etc.). In constrained problems, complexities arise in GAs due to how the genetic operators direct the search. It is very likely that a small change to a feasible solution will lead to an infeasible one. (Ruiz-Andino, et al., 2000:353).

One approach to dealing with constraints is to modify the solution representation itself so as not to allow the creation of infeasible solutions. Repair algorithms or decoders are special operators that avoid the construction of illegal solutions. They may work reasonably well but are highly problem specific and may be computationally intensive to run. Additionally, they may work against the inherent search properties of the GA and may be difficult to implement (Ruiz-Andino, et al., 2000:353).

Penalty functions that reduce the fitness of infeasible solutions are more popular (Kundu, 1995:96, Deb, 1999a:14) but also problem specific. Penalty functions may be linear, quadratic, logarithmic, etc. functions of the deviation of the constraints and/or the number of violated constraints. Although successfully used by many researchers, the performance of GAs will depend upon the choice of constraint parameter values used. To prevent the emphasis of a particular constraint and thereby restrict the search, different penalty parameters should be used with different objective functions (Deb,

1998:7, Gray, et al., 1997). Deb describes a parameterless penalty function that is used with a size-2 tournament selection operator (Deb, 1999a:15):

Given a single objective function, $f(x)$, and the maximization inequality

$$g_j(x) \geq 0, \quad j = 1, 2, \dots, J \quad (2.12)$$

the fitness, $F(x)$, of any solution is defined as follows:

$$F(x) = \begin{cases} f(x), & \text{if } g_j(x) \geq 0, \quad \forall j \in J, \\ f_{\max} + \sum_{j=1}^J g_j(x), & \text{otherwise} \end{cases} \quad (2.13)$$

where f_{\max} is the maximum function value of all feasible solutions in the population.

In a tournament between an infeasible solution and feasible solution, it can be seen from equation (2.13) that the feasible solution always has a better fitness than the infeasible ones. If both solutions are feasible, their assessment is based on their respective objective function values. If both solutions are infeasible, then the assessment is made based on the amount of the constraint violations. No penalty parameter need be used since pairwise comparison of the infeasible solutions does not depend on their exact fitness values (Deb, 1999a:15).

Multiobjective GAs. The basic operation of the single objective GA in Figure 3 must be enhanced to evaluate solutions with multiple fitnesses (objectives). Researchers have responded with a number of ways to judge the overall fitness of the solutions. Van Veldhuizen's recent research served to classify known multiobjective EAs on the basis of

the role of the decision maker in the process (1999:A-1). In *a priori* techniques, the decision maker makes his or her preferences known at the beginning of the process, resulting in a single compromise solution, e.g. lexicographic (ordering), linear and non-linear combination, or goal programming.

In *a posteriori*, the Pareto optimal set is generated for the decision maker who then makes his or her preferences known by selecting a solution from the set. This technique is notable in that the resultant solution set is independent of the decision maker's preference and, assuming no change in the problem environment, new solution sets would not need to be generated for different decision makers.

Progressive techniques allow the decision maker to interact and provide preference information during the process. These techniques require a high degree of participation from the decision maker and generally make use of both *a priori* and *a posteriori* techniques.

A large number of methods for judging overall fitness use an objective-aggregation approach and fall in the category of *a priori* techniques. The different fitness values are weighted and summed according to the decision maker's preference for them. However, this is very subjective and is difficult to do accurately, especially when the interplay between non-commensurate objectives is not well understood (Chipperfield, et al., 2000, Shaw, 1998). The search space is inextricably linked to the weightings, thus a single inaccurate weight may cause a GA converge to an undesirable front.

The predominant approach to solving MOEAs is to use the concept of Pareto dominance, as defined in Appendix A, in the selection operator (Deb, 1999:4). Van Veldhuizen notes that "the sheer number of Pareto sampling approaches indicates many

researchers see merit in the basic methodology” (1999:3-10). Pareto dominance allows all nondominated solutions to have the same preference, resulting in a set of nondominated solutions for which the population-based EA is particularly well suited to handle. Pareto dominance approaches produce as their end product nondominated sets of solutions and so are well suited for use in the a posteriori mode. The following algorithms are among the most often cited and *copied* contemporary MOEAs that use Pareto dominance:

- Multiobjective GA (MOGA) (Fonseca and Fleming, 1993)
- Niched Pareto GA (NPGA) (Horn, et al., 1993)
- Non-dominated Sorting GA (NSGA) (Srinivas and Deb, 1995)
- Strength Pareto EA (Zitzler and Thiele, 1998)
- NSGA-II (Deb, et al., 2000)

In order to improve the explorative and exploitative properties of their respective algorithms, researchers have used more complex selection operators, such as ranking, sharing, niching, elitist, and domination tournaments (Zydallis, et al., 1999:2).

MOGA and NSGA use variations on Goldberg’s nondominated sorting procedure. The basic operation of this procedure is to rank solutions in nondominated order with the best solutions being the least dominated. These fittest solutions are given higher probabilities of producing more offspring (Bentley and Wakefield, 1999:7). The MOGA checks the population and assigns a rank of 1 to all nondominated solutions. Each of the other solutions is ranked based on the number of solutions that dominate it (Deb, 1999b:5). Fitness is assigned based on linear or exponential interpolation (Van Veldhuizen, 1999:3-22).

NSGA also checks the population and assigns a rank of 1 to all nondominated solutions, forming what it calls the *first level* of non-domination. The first level is removed from consideration and proceeds on the rest of the population in the same way, resulting in 1 to n domination levels, n being the population size. All first level solutions receive a fitness equal to the population size. The other levels receive a *dummy* fitness that is smaller than the smallest shared fitness of the preceding level.

Fitness *sharing* was suggested by Goldberg allow for solutions with identical fitness along different parts of the front, thereby helping the population to be distributed along the front (Horn, et al.,1993:4). The number of neighboring solutions along the front, referred to as a *niche*, are used to selectively reduce the fitness of high niche count solutions, thus increasing pressure toward a uniform Pareto front distribution (Horn, et al., 1993:8). A sharing parameter, σ_{share} , is a defined maximum distance within which any solution constitutes as belonging to a neighborhood. Solutions within σ_{share} of each other reduce each others fitness.

To perform domination ranking, NPGA uses domination tournaments of size two (Horn, et al., 1993: 6). The tournament procedure selects two solutions at random and each of them competes against a comparison set of solutions, t_{dom} , that are also selected at random from the population. When one solution is dominated and the other is not, the latter is selected. When competing solutions are either both dominated or both nondominated, sharing determines the winner. NPGA implements sharing in a different manner. Rather than reducing the fitness of high niche count solutions, the winner is declared based on the solution with the smallest niche count (Horn, et al., 1993:9). While

MOGA, NSGA, and NPGA all require explicit values for σ_{share} , NPGA also requires the same for t_{dom} .

Elitist selection ensures that the best solutions are retained in the next generation (Van Veldhuizen, 1999:A-25). SPEA uses an elitist selection with nondomination (Deb, 1999b:6). The algorithm maintains a secondary population that is the current Pareto optimal set. This population is combined with the current population and nondominated comparisons are performed on the whole. Nondominated solutions are assigned a fitness based on the number of solutions they dominate. Preference is given to

1. nondominated solutions that dominate more solutions in the combined population, and
2. dominated solutions that are dominated by more solutions in the combined population.

The preference rules are meant to check premature convergence by preventing large numbers of good solutions from being carried over from one generation to the next (Deb, 1998:26). However, rather than having equal preference for all nondominated solutions, SPEA is biased in favor of nondominated solutions that dominate more solutions than others (Van Veldhuizen, 1999:3-20).

There is another approach that steps further away from the traditional GA: the Messy GA (mGA). The mGA abandons fixed length bit strings and so-called *neat* operator, crossover, in favor of variable-length based representations, and a gene reordering operator called *cut-n-splice*. The pseudo code for Goldberg's, et al. mGA is presented in Figure 5.

```

Start mGA
  // loop for user defined number of eras
  while not done do
    // perform Phase 1: Partially Enumerative Initialization
    // evaluate individual fitness for entire population
    // start Phase 2: Primordial Phase
    // loop for user defined number of generations
    while not done do
      // increment the generation counter
       $g = g + 1$ ;
      // perform Tournament Thresholding Selection
      // test for appropriate number of elapsed
      generations,  $g^*$ 
      if  $g = g^*$ ;
        reduce population size;
        // reset  $g$ 
         $g = 0$ ;
      end if
    end do
    // end Primordial Phase
    // start Phase 3: Juxtapositional Phase
    // loop for specified number of generations
    while not done do
      // increment the generation counter
       $t = t + 1$ ;
      // perform cut-n-splice
      // evaluate individual fitness for entire population
      // perform Tournament Thresholding Selection
    end do
    // end Juxtapositional Phase
    // update competitive template
  end do
end mGA

```

Figure 5. Pseudo Code for a Messy Genetic Algorithm (Van Veldhuizen, 1999:4-5)

A GA has difficulty when the genes are not ordered properly. According to the Schema Theorem, these longer length schemata have a higher probability of being destroyed by crossover and mutation. This linkage problem leads to what is known as *deception*, where poorly ordered schemata lead the GA away from the global optimum.

This is illustrated in the following example by Goldberg, et al., (1989:508). Given that $(00*****)$ and $(*****00)$ are highly fit schemata of an optimal point (11111111) , and that the schema $(00****00)$ is much less fit than the building block $(11****11)$, the GA will, with high probability, destroy the longer length building block and converge to a less than optimal point.

Nature allows individuals to carry redundant information such as multiple copies of genes and paired chromosomes. Messy genetic algorithms copy this by allowing redundant or even contradictory genes (Goldberg, et al., 1989:501). To allow the reordering of genes, each gene is a pair of integers that represents the name and value of the gene, respectively. For example, in messy representation, two strings P_1 and P_2 are

$$P_1 = ((3,1) (1,0) (3,0) (2,1) (1,0)) \quad (2.14)$$

and

$$P_2 = ((4,1) (2,0) (3,0) (2,1)) \quad (2.15)$$

Both P_1 and P_2 are valid despite under-specification by P_1 in bits four and five, and over-specification by P_1 in bit 3 and by P_2 in bit 2.

The traditional crossover operator is replaced by the *cut-n-splice* operator. The name of this operator is indicative of what it does to bit strings. The position of cuts can be chosen independently for both parents. After cutting, partial strings are spliced in a random order. To illustrate, cut-n-splice is performed on P_1 and P_2 in Figure 6.

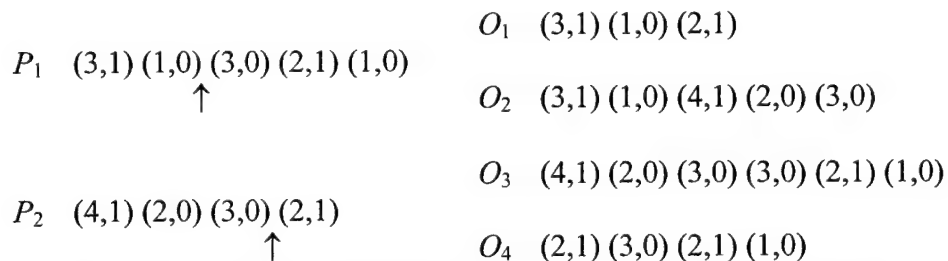


Figure 6. Example of Cut-n-Splice Operation (Hoffman, 1997)

Evaluation of variable lengths strings is problematic since under- and over-specified strings must have their lengths changed to fit with the objective function. Goldberg, et al., settled on a simple first-come, first-served process to handle over-specification (1989:501). Since this method does not rely on bitwise fitness for its choice, it is not biased to toward deceptive schemata. Goldberg, et al., successfully handle under-specification through their use of *competitive templates* that fill in the unspecified bits in an under-specified string (1989:521). A competitive template is initialized randomly and used in the first era. Thereafter, the best solution in the current era is used as the competitive template for the next era, and so on. A competitive template that is itself a locally optimal solution to a problem “accentuates salient building blocks” by ensuring that their fitness is better than that of the template (Goldberg et al., 1990:417).

An mGA proceeds in two phases. Prior to the first phase, the population is initialized so that it completely enumerates building blocks of a given length. This process is referred to by Goldberg, et al. as *Partially Enumerative Initialization* (1990:505). The population sized is determined based on the highest order k deceptive string expected in the problem:

$$n = 2^k \binom{l}{k} \quad (2.16)$$

where l is the length of the chromosome (Goldberg, et al., 1989:505).

In the *primordial* phase, tournament selection is performed on successive populations “to create an enriched population of building blocks whose combination will create optimal or very near optimal strings” (Goldberg, et al., 1989:505). To avoid an

apples and oranges comparison of substrings that do not refer to the same subfunction, the mGA only compares substrings that are similar to each other to an extent that is defined by a *threshold* number of genes in common. The threshold parameter is defined by Goldberg, et al., to be

$$\theta = \frac{l_1 l_2}{l} \quad (2.17)$$

where l is the chromosome length and l_1 and l_2 are the respective substring lengths (1989:426). The number of substrings to check against θ is defined by a *shuffle number*, n_{sh} , equal to the chromosome length (Goldberg, et al., 1989:427).

As the primordial phase proceeds, the population size is reduced roughly in half by selection at specified intervals since only the better building blocks need to be maintained. This phase represents a key difference between a mGA, which explicitly and directly manipulates building blocks, and other EAs which settle for implicitly manipulating building blocks.

At the conclusion of the primordial phase, the juxtapositional phase proceeds as a traditional GA would on a fixed population size, albeit using cut-n-splice instead of crossover to lengthen the substrings. Thresholding is also used in the selection operator. In essence, an mGA tries to gather information on building block relationships first, then searches for better solutions (Kargupta, no date).

In his PhD dissertation, David Van Veldhuizen extended the mGA to handle MOPs and developed the Multi-Objective Messy Genetic Algorithm (MOMGA) (1999). There is a competitive template for each objective that is at first randomly initialized and then updated with the previous era's best individual for that objective (Zydallis, et al.,

2001:4). Like Horn, et al., Van Veldhuizen augmented the tournament selection operator with a niching strategy to increase domination pressure (1999:4-14). As with NPGA, the MOMGA requires explicit values for σ_{share} and t_{dom} to control domination pressure. The MOMGA uses Fonseca's suggested method to determine σ_{share} :

$$N = \frac{\prod_{i=1}^k (\Delta_i + \sigma_{share}) - \prod_{i=1}^k \Delta_i}{\sigma_{share}^k} \quad (2.18)$$

where N is the number of individuals in the population, Δ_i is the difference between the maximum and minimum objective values in dimension i , and k is the number of distinct MOP objectives (Van Veldhuizen, 1999:6-11).

The MOMGA also maintains and updates a list of known Pareto optimal solutions P_{known} with Pareto optimal solutions from current generation $P_{current}$ (Van Veldhuizen, 1999:4-17). Since dominance determination is at worst an n^2 algorithm, n being the list cardinality, it is done on P_{known} at the termination of the program to prevent the MOMGA by being bogged down.

As shown by equation (2.16), the initial population grows exponentially as the building block size k is increased, creating a computational bottleneck, i.e. $O(l^k)$. Zydallis, et al. reduce this bottleneck using a probabilistic approach to initialize the population (2001:5). *Probabilistically Complete Initialization* (PCI) creates a controlled number of building blocks of size k . *Building Block Filtering* (BBF), which replaces the Primordial phase, alternately reduces string lengths by randomly deleting bits from the strings and performs selection on the strings. This continues according to a user specified schedule of alternations until the strings are of length k . The Juxtapositional phase

proceeds as before. This approach probabilistically ensures that all of the best building blocks are in the initial population and results in initial population growth on order of the initial string length— $O(l)$ (Goldberg, et al., 1993:7).

Using these ideas, Zydallis, et al. modified original MOMGA, creating a *multiobjective fast messy GA* (MOMGA-II). Their research shows the MOMGA-II to be more efficient than the MOMGA while reutilizing much of the same code. The MOMGA-II was also applied to the same test suite as the original MOMGA, achieving similar results but with fewer juxtapositional generations (2001:10).

Summary

To answer the first research question, we began by defining what a multiple objective programming problem is, what it means to be globally optimal, the concept of Pareto dominance, and introduced classical approaches to solving MOPs. We then presented modern methods for solving MOPs in terms of two approaches: local search in objective space and population-based.

The next chapter answers the second and third research questions by presenting a multiobjective model formulation and solution methodology for the research problem.

III. Methodology

Introduction

The previous chapter was directed at the first research question, which asks for a review of MOP methodologies. The intent of this chapter is to answer the second and third research questions:

- What are the forms of the decision and objective spaces?
- How is the selected MOP methodology evaluated?

This chapter also defines the experimental methodology that is used to answer the fourth question: evaluate the MOP approach used to solve the research problem.

This chapter begins with the research problem model formulation that is based on the ALP Pilot Problem (Swartz, 1999). The next section presents the MOP formulation and describes the construction of the research problem's decision variables, objective functions, and constraints. This is followed by a description of the target MOP used in the model. Next, the motivation for selection of a specific MOP methodology and objectives for its evaluation are discussed. The last two sections present evaluation metrics and the solution methodology used for this research.

Model Formulation

This research problem is modeled on the ALP Pilot Problem presented by Stephen Swartz (1999). The reader is referred to Appendix B for the background information used to construct the model.

The MOP Processing Model is depicted in Figure 7. The centerpiece of the model is the MOP Tool. The MOP Tool is to be an application of a MOP solution methodology selected from the literature. The inputs to the MOP tool allow for the MOP formulation discussed in the next section.

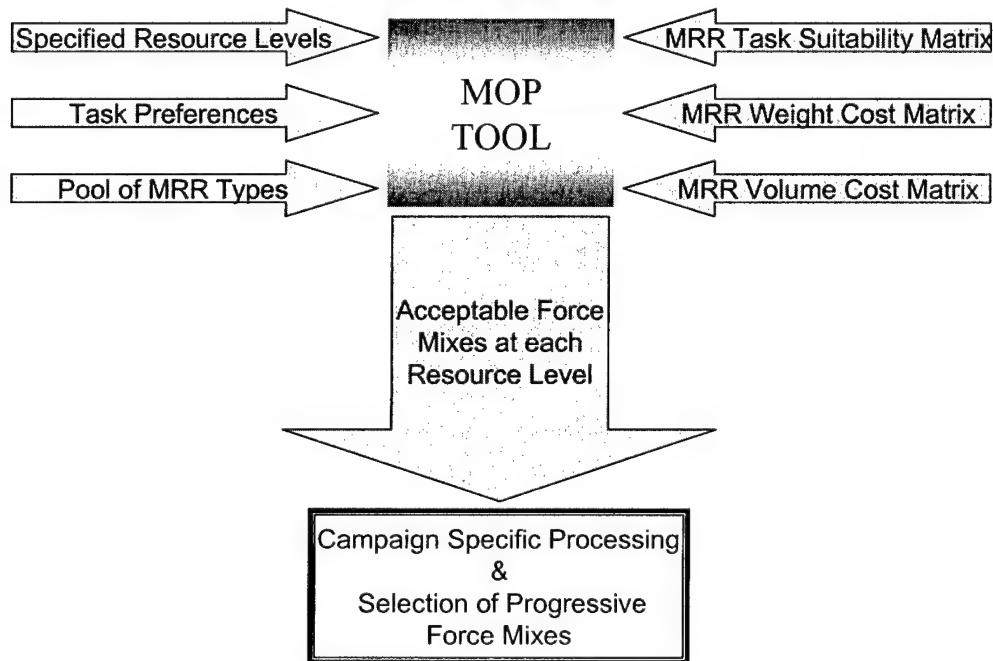


Figure 7. MOP Processing Model

The output requirement of the model is to present to the war planner a Pareto optimal set of force mixes from which to select the desired force mix. Using some decision making methodology, the planner can choose the desired Mission Ready Resource (MRR) set from each Pareto optimal set associated with an inflection point on the task preference vector (as shown in Figure 12 of Appendix B). This piecewise solution represents the decision maker's preferred MRR sets for a given combat capability.

These assumptions imply that a solution set produced at a given resource level is a subset of solution sets at greater resource levels. This must be reflected in the piecewise solution and is accomplished by proceeding along the preference vector from the origin to the last resource level or vice versa. This is illustrated by the three force mix sets in Figure 8. For purposes of illustration, each set is considered nondominated. There are nine possible force mix threads that progress from the lowest to highest specified resource levels: {1, 2, 5}, {1, 2, 6}, {1, 2, 7}, {1, 3, 5}, {1, 3, 6}, {1, 3, 7}, {1, 4, 5}, {1, 4, 6}, and {1, 4, 7}. However, it is seen by inspection that only three of these meet the subset criteria: {1, 2, 7}, {1, 3, 6}, and {1, 3, 7}.

“Thread of Progression”

Resource Level = 60					
<i>Force Mix 5</i>		<i>Force Mix 6</i>		<i>Force Mix 7</i>	
F_A	F_B	F_A	F_B	F_A	F_B
13	47	15	30	30	30
Resource Level = 27					
<i>Force Mix 2</i>		<i>Force Mix 3</i>		<i>Force Mix 4</i>	
F_A	F_B	F_A	F_B	F_A	F_B
16	11	14	13	5	21
Resource Level = 12					
<i>Force Mix 1</i>					
F_A	F_B				
6	6				

Figure 8. Thread of Progression

It is tempting to handle this requirement using a series of constraints. To illustrate, we select a progression direction that begins with Resource Level 12. We then

choose the upper bound for each MRR type at that resource level to be the starting point, or lower bound, for the next resource level. This leaves $\{1, 2\}$ and $\{1, 3\}$ as feasible threads. Now the upper bound for each MRR type at this level is the starting point for the next level. This leaves only $\{1, 2, 7\}$ and $\{1, 3, 7\}$. Although $\{1, 3, 6\}$ meets the subset requirement, it is deemed infeasible. Using the lower bound as the starting point for the next level allows the infeasible threads $\{1, 2, 5\}$ and $\{1, 2, 6\}$. The same kind of problem exists when starting from the highest resource level and progressing downward.

It is my opinion that the best way to handle the construction of force mix threads is through a post-processing algorithm. The algorithm operates on level-wise nondominated sets of force mixes and, starting at the lowest (or highest) resource level, constructs threads iteratively, taking into consideration all possible feasible threads. Due to time constraints, construction of this algorithm will not be undertaken in this research.

MOP Formulation

Given m tasks and n MRR types, the solution set is an $m \times n$ matrix. A matrix element is a decision variable, $x_{i,j}$, that represents the number of MRRs of type j allocated to tasks of type i . Assuming that each daily task is satisfied by exactly one MRR, and that no interactions exist between differing MRR types, then the suitability, S , for all MRRs is defined by

$$S = \sum_{j=1}^n \sum_{i=1}^m \alpha_{i,j} x_{i,j} \quad (3.1)$$

where $\alpha_{i,j}$ is the suitability of MRR j for Task i and $x_{i,j}$ is the number of MRRs j allocated to task type i .

The requirement that all tasks $i = 1, \dots, n$ must be satisfied at a particular resource level (RL) k is:

$$RLtask_{k,i} = \sum_{j=1}^n x_{i,j} \quad (3.2)$$

Since the desired capability for a task is set by the decision maker and defined to be static, the left-hand side of equation (3.2) is an equality constraint.

The requirement that all MRR types $j = 1, \dots, n$ do not exceed their available number at a particular resource level k is

$$RLmrr_{j,k} \geq \sum_{i=1}^m x_{i,j,k} \quad (3.3)$$

In this problem, the decision variables are allowed to take on any non-negative integer value so long as they do not exceed the specified resource level. Therefore, the left-hand side of equation (3.3) is an inequality constraint.

The maximum number of sorties per day for a particular asset, A , is given by its *turn rate*, t . For a quantity d of asset A , the total turn rate is

$$TTR_A = (d_A)(turn\ rate_A) \quad (3.4)$$

Given that A has P configurations corresponding to P MRR types, the upper bound for any combination of the P MRR types is

$$TTR_A \geq \sum_{r=1}^P \sum_{i=1}^m x_{i,p_r} \quad (3.5)$$

For example, let the number of tasks be one, and let $P = \{1, 3, 4\}$ be the set of MRR types that correspond to asset A . If the number of A is one and the turn rate of A is two, then following decision variable values are possible:

$x_{l,1}$	$x_{l,3}$	$x_{l,4}$
1	1	0
1	0	1
0	1	1
2	0	0
0	2	0
0	0	2
1	0	0
0	1	0
0	0	1
0	0	0

Mathematically, this table is represented as

$$\sum_{r=1}^3 \sum_{i=1}^1 x_{i,p_r} \leq TTR_A$$

$$x_{1,1} + x_{1,3} + x_{1,4} \leq 2$$

It is difficult to determine what the actual logistical footprint is for a given asset

set. At the very least, it is clear that for each additional asset deployed, there is a

corresponding increase in lift cost for additional resources, e.g. fuel, munitions, etc.

Research conducted by Matt Goddard suggests that, for F-16s, the relationship of asset

quantity to lift resource consumption is linear (2001). Assuming that lift consumption is

linear and without interaction, the weight consumption, W , and volume consumption, V ,

for all MRRs are

$$W = \sum_{j=1}^n \sum_{i=1}^m \beta_j x_{i,j} \quad (3.6)$$

and

$$V = \sum_{j=1}^n \sum_{i=1}^m \lambda_j x_{i,j} \quad (3.7)$$

where β_j and λ_j are the weight and volume consumed by a single MRR j .

The form of the *suitability maximizing / lift minimizing MOP* with A asset types, m tasks, n MRR types, at a resource level k , and decision variables $\{x_{1,1}, x_{1,j}, \dots, x_{m,n}\}$ is

maximize:

$$S = \sum_{j=1}^n \sum_{i=1}^m \alpha_{i,j} x_{i,j}$$

minimize:

$$W = \sum_{j=1}^n \sum_{i=1}^m \beta_j x_{i,j}$$

$$V = \sum_{j=1}^n \sum_{i=1}^m \lambda_j x_{i,j}$$

subject to

$$\sum_{j=1}^n x_{i,j} = RLtask_{k,i} \text{ for } i = 1 \text{ to } m \quad (3.8)$$

$$\sum_{i=1}^m x_{i,j} \leq RLmrr_{k,j} \text{ for } j = 1 \text{ to } n \quad (3.9)$$

$$\sum_{r=1}^{P_a} \sum_{i=1}^m x_{i,p_r} \leq TTR_a \text{ for } a = 1 \text{ to } A, \text{ and} \quad (3.10)$$

P_a = number MRR types for a

$\{x_{1,1}, x_{1,j}, \dots, x_{m,n}\}$ are non-negative integers

The number of constraints resulting from equation (3.8) is equal to the number of tasks. These constraints ensure that the total number of sorties for Task i is exactly the desired capability at that resource level. The maximum value for any decision variable is found by using equation (3.8) and allocating all task capability to one MRR type. This

information is important to the MOP programmer who must allocate computer memory to hold the value for each decision variable.

The number of constraints resulting from equation (3.9) is equal to the number of MRR types. These constraints ensure that no MRR type can be allocated a number of sorties that exceeds the given resource level. These constraints are also used when there are restrictions on the available number of any MRR type, e.g. attrition or changes in asset turn rate. It is important to note that each constraint refers to a single MRR type.

Target MOP

The MOP Tool inputs in Tables 2 through 5 provide linkage between this thesis and concurrent ALP research, and create a search space large enough to serve as a reasonable test of the MOP Tool's utility to the problem. The inputs are completely notional but not entirely arbitrary.

The number of tasks in Table 1 and MRR types in Table 5, along with their task suitabilities, are set to provide proper input to concurrent research (Filcek, 2001). The suitabilities reflect notional but reasonable values that clearly differentiate the MRR types. The same can be said for the lift consumption values in Table 5.

To keep the number of task capability decisions by the decision maker at a reasonable level, five resource levels in Table 2 were specified, equating to 15 separate task preference decisions. These preferences are reflected in Table 3. When the ratios are applied to their respective resource level values, the result is the capability matrix in Table 4. The values are rounded to a whole number so that the sum across tasks is equal to the resource level.

The values of the resource levels were chosen to create solution spaces of increasing size. Given three tasks and five MRR types and a resource level of 300 sorties per day, the worst case number of possible force mixes is approximately 9.72×10^{19} (by equation (B.2)).

For the target MOP, it is assumed that there is no restriction on the available number of any MRR type; the combined lift will not exceed the maximum available lift; no attrition; and that each asset has one associated MRR type, i.e. one sortie per day. These simplifying assumptions are made to meet research time constraints, but also allow this groundbreaking research an opportunity to explore the basic problem complexity.

Table 1. Tasks

INDEX	NOMENCLATURE
1	Air-to-Air (AA)
2	Air-to-Ground (AG)
3	Precision Bombing (PB)

Table 2. Resource Levels (RLs)

INDEX	RL (sorties per day)
1	16
2	32
3	75
4	150
5	300

Table 3. Desired Task Capability Ratios

INDEX	PERCENT TO TASK		
	<i>AA</i>	<i>AG</i>	<i>PB</i>
1	60	30	10
2	30	60	10
3	25	60	15
4	20	50	30
5	20	30	50

Table 4. Desired Capability Matrix

INDEX	TASK (sorties per day)			DECISION SPACE CARDINALITY
	<i>AA</i>	<i>AG</i>	<i>PB</i>	
1	10	5	1	630,630
2	10	20	2	159,549,390
3	19	45	11	$\approx 2.56 \times 10^{12}$
4	30	75	45	$\approx 1.48 \times 10^{16}$
5	60	90	150	$\approx 4.37 \times 10^{19}$

Table 5. Task Suitability / Lift Consumption Matrix

INDEX	MRR Type	TASK SUITABILITY			LIFT CONSUMPTION	
		<i>AA</i>	<i>AG</i>	<i>PB</i>	<i>WEIGHT</i> (short tons)	<i>VOLUME</i> (cubic feet)
1	F_A	0.800	0.400	0.001	20.2	1650.0
2	F_B	0.300	0.800	0.001	28.5	2475.0
3	F_C	0.600	0.600	0.100	35.7	2887.5
4	B_1	0.001	0.001	0.800	19.9	1705.0
5	B_2	0.001	0.001	0.400	22.5	2200.0

The complete MOP formulation is as follows:

Decision variables: Number of MRR j assigned to Task $i = \{x_{1,1} \dots x_{i,j}\}$

Maximize:

$$S = 0.8x_{1,1} + 0.3x_{1,2} + 0.6x_{1,3} + 0.001x_{1,4} + 0.001x_{1,5} + 0.4x_{2,1} + 0.8x_{2,2} + 0.6x_{2,3} + 0.001x_{2,4} + 0.001x_{2,5} + 0.001x_{3,1} + 0.001x_{3,2} + 0.1x_{3,3} + 0.8x_{3,4} + 0.4x_{3,5} \quad (3.11)$$

Minimize:

$$W = 20.2(x_{1,1} + x_{2,1} + x_{3,1}) + 28.5(x_{1,2} + x_{2,2} + x_{3,2}) + 35.7(x_{1,3} + x_{2,3} + x_{3,3}) + 19.9(x_{1,4} + x_{2,4} + x_{3,4}) + 22.5(x_{1,5} + x_{2,5} + x_{3,5}) \quad (3.12)$$

$$V = 1650(x_{1,1} + x_{2,1} + x_{3,1}) + 2475(x_{1,2} + x_{2,2} + x_{3,2}) + 2887.5(x_{1,3} + x_{2,3} + x_{3,3}) + 1705(x_{1,4} + x_{2,4} + x_{3,4}) + 2200(x_{1,5} + x_{2,5} + x_{3,5}) \quad (3.13)$$

Subject to:

$$x_{1,1}, \dots, x_{3,5} \geq 0 \quad (3.14)$$

$$\{x_{1,1}, \dots, x_{3,5}\} \in I \quad (3.15)$$

$$x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} + x_{1,5} = RLtask_{m,1} \quad (3.16)$$

$$x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} + x_{2,5} = RLtask_{m,2} \quad (3.17)$$

$$x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} + x_{3,5} = RLtask_{m,3} \quad (3.18)$$

$$x_{1,1} + x_{2,1} + x_{3,1} \leq RLmrr_{m,1} \quad (3.19)$$

$$x_{1,2} + x_{2,2} + x_{3,2} \leq RLmrr_{m,2} \quad (3.20)$$

$$x_{1,3} + x_{2,3} + x_{3,3} \leq RLmrr_{m,3} \quad (3.21)$$

$$x_{1,4} + x_{2,4} + x_{3,4} \leq RLmrr_{m,4} \quad (3.22)$$

$$x_{1,5} + x_{2,5} + x_{3,5} \leq RLmrr_{m,5} \quad (3.23)$$

where m is the Resource Level index for the current problem.

Motivation and Objectives

The aim of this research is to select and evaluate a MOP tool that outputs to a post-processor an acceptable series of resource level constrained force mixes in a reasonable amount of time, and is scalable to more realistic problem sizes. In their basic forms, simulated annealing and genetic algorithms are easy to understand and implement. Both methods can show convergence, albeit slowly, when good solutions now are better than great solutions later.

While many candidate approaches exist in the literature, an evolutionary-based approach is ideally suited to the search for nondominated sets of solutions, particularly for multidimensional problem domains and large search spaces. The balance between exploration pressure and exploitation pressure can be controlled nearly independently in GA, allowing flexibility in design (Deb, 1999a:11). In addition, GA populations and operators can be parallelized, allowing scaling to large problems by using multiple processors to reduce overall computational time.

The choice to design an MOEA for the problem domain or to modify an existing MOEA to incorporate problem domain knowledge is a practical one. All methods rely heavily on computing skills for implementation, and the time allotted for this research is limited. While a simple GA can be designed for the problem at hand, clever enhancements are needed to increase the efficiency and effectiveness of the algorithm and make it truly useful. The contemporary MOEAs cited in the literature review are well designed, based on modern MOEA theory, and have found use in many applications (Van Veldhuizen, 1999). Van Veldhuizen identified quantitative metrics to objectively compare four such MOEAs in terms of their efficiency and effectiveness: MOGA,

NSGA, NPGA, and MOMGA. The reader is referred to Chapter II for an overview of these MOEAs. Using a carefully designed test suite of MOPs that emphasized certain genotypical and phenotypical characteristics from throughout the MOP domain (concave, convex, connected, non-connected, scalable, uniformity, and non-uniformity), he concluded that the MOMGA compared favorably with the other MOEAs and surpassed the effectiveness of several of them (1999: 5-5, 5-7, 7-23, Zydallis, et al., 2001:1,14). These results validated the use of explicit building block MOEAs in the MOP domain. A similar experiment by Zydallis, et al., pitted the MOMGA-II against the same MOEAs and concluded that the MOMGA-II achieves results similar to that of the MOMGA, but in a more efficient manner (2001:14).

The algorithm selected as the MOP tool is the MOMGA-II. Van Veldhuizen comments that “although no guarantor of continued success, any search algorithm giving effective and efficient results over the test suite might be easily modified to target specific problems” (1999:5-7). MOMGA-II experimental results on pedagogical problems are encouraging and its architecture is readily modifiable for the test problem’s high dimensionality and heterogeneous objectives. The Pareto dominance routine is simpler to implement when the objectives are either all minimization or all maximization. Therefore, the target MOP’s first objective is to minimize the multiplicative inverse of equation (3.11).

MOMGA-II use of memory is illustrated in Figure 9. Memory usage increases linearly with string length, the slope of which is dependent upon the population size. The maximum problem length for the target problem is 120 bits (from Table 4), requiring

only 1.36 kB per population member. A suitably equipped PC can accommodate much larger problems.

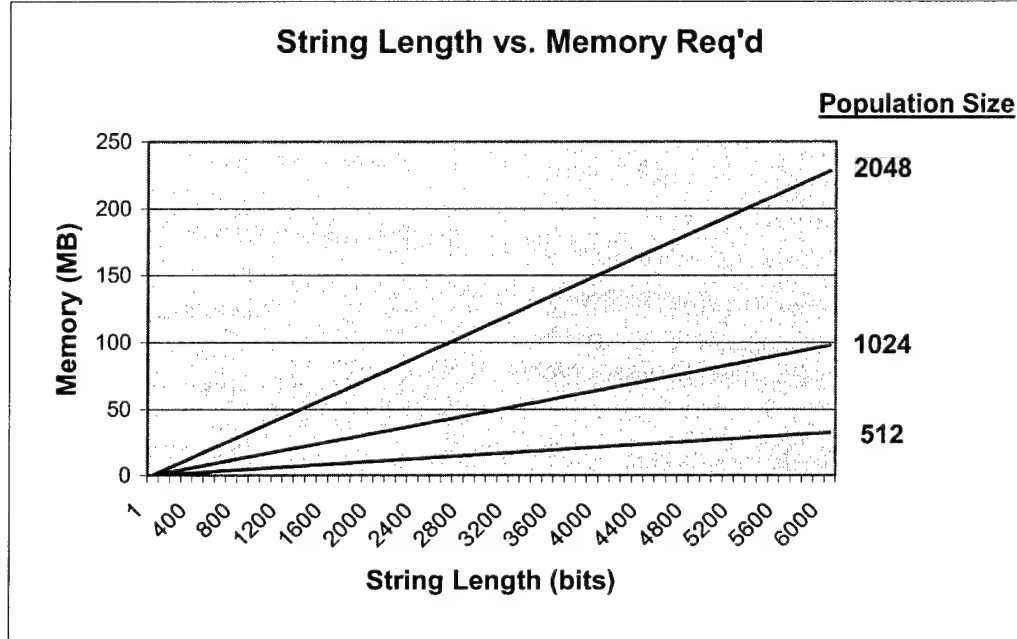


Figure 9. MOMGA-II String Length vs. Memory Required

The first objective is to compare P_{true} and PF_{true} from the enumerative approach to P_{known} and PF_{known} from the MOMGA-II. Since enormous computational resources are required to enumerate P_{true} and PF_{true} beyond index 1 from Table 4, all absolute comparisons can only be made at a single specified resource level. To explore the performance of the algorithm on the target MOP, an analysis of selected MOMGA-II operating parameters can determine the individual impact of those parameters on the algorithm's effectiveness.

The *No Free Lunch* theorem tells us that on average, all search algorithms perform equally well or equally bad over all problems (Wolpert and Macready, 1996:2). Therefore, applying a search algorithm without regard for problem domain knowledge

can lead to performance no better than a random search. Two aspects from the research problem domain to consider are interrelationships between decision variables, decision variable string length, and application of side constraints.

Each decision variable is a member of two groups: Task Number and MRR Type. The strongest relationship is the Task Number, as evidenced by equations (B.1) and (B.2). The strict equality of the decision maker's task preferences means that for one decision variable to increase, one or more others from the same Task Number group must decrease. This is also true for MRR types, but to a far lower extent. The best genotypic representation for related decision variables would be one that fosters this relationship. The Building Block Hypothesis suggests a representation that encodes them in close proximity to one another. This idea can be used in the primordial phase of the MOMGA-II by using building blocks of size equal to the bitstring representation of the decision variables.

The traditional binary encoding of decision variables is used for the target MOP. Since the decision space of target MOP is defined to be in I , the MOMGA-II can be designed to use either a standard bit string length for each decision variable or bit string lengths that depend on the size of each decision variable. While the latter economizes on the memory required for each vector solution, the destruction of information by recombination may be biased toward the larger representations (assuming that the random number generator output is normally distributed). Therefore, the representation length for each decision variable is defined to be that for the largest decision variable.

The algorithm designer typically has two choices when using side constraints to ensure a feasible region. If the constraints are applied within the algorithmic process in

order to incorporate some problem domain knowledge, solutions that are found to be infeasible are penalized or thrown out, diminishing their impact on the search. This works against a GA, which uses the fitness landscape information carried by populations to direct the search, and risks premature convergence. If the constraints are applied a posteriori, information from the entire fitness landscape is potentially available to the algorithm. The resultant unconstrained nondominated front is then culled of infeasible members. The comparatively larger search space may require a greater number of fitness evaluations to converge to the Pareto front. It may also be that the unconstrained PF_{true} does not contain any point of the constrained PF_{true} .

Since the best approach is unknown, and given limited time to complete this research, the second objective is to demonstrate the ability of the applied algorithm to produce level-wise nondominated force mix sets as defined by the model formulation. Since the constraints in equations (3.8) and (3.9) are not handled explicitly by the algorithm, the decision variable domains must be specified explicitly using the bitstring representing each decision variable. This version of the MOMGA-II will use a standard length binary representation for all decision variables.

Preliminary runs of the MOMGA-II on the target MOP using index 1 of Table 4 and the parameters listed in Table 7 generated no feasible solutions. Referring to Figure 10, the objective space of the target MOP at the lowest resource level defines a *point mass feasible field*. This is a particularly difficult problem for any optimization method. When an algorithm finds itself at any feasible point, it is surrounded by infeasible points of varying density. There is no guarantee that the next move off of a feasible point will result in another feasible point.

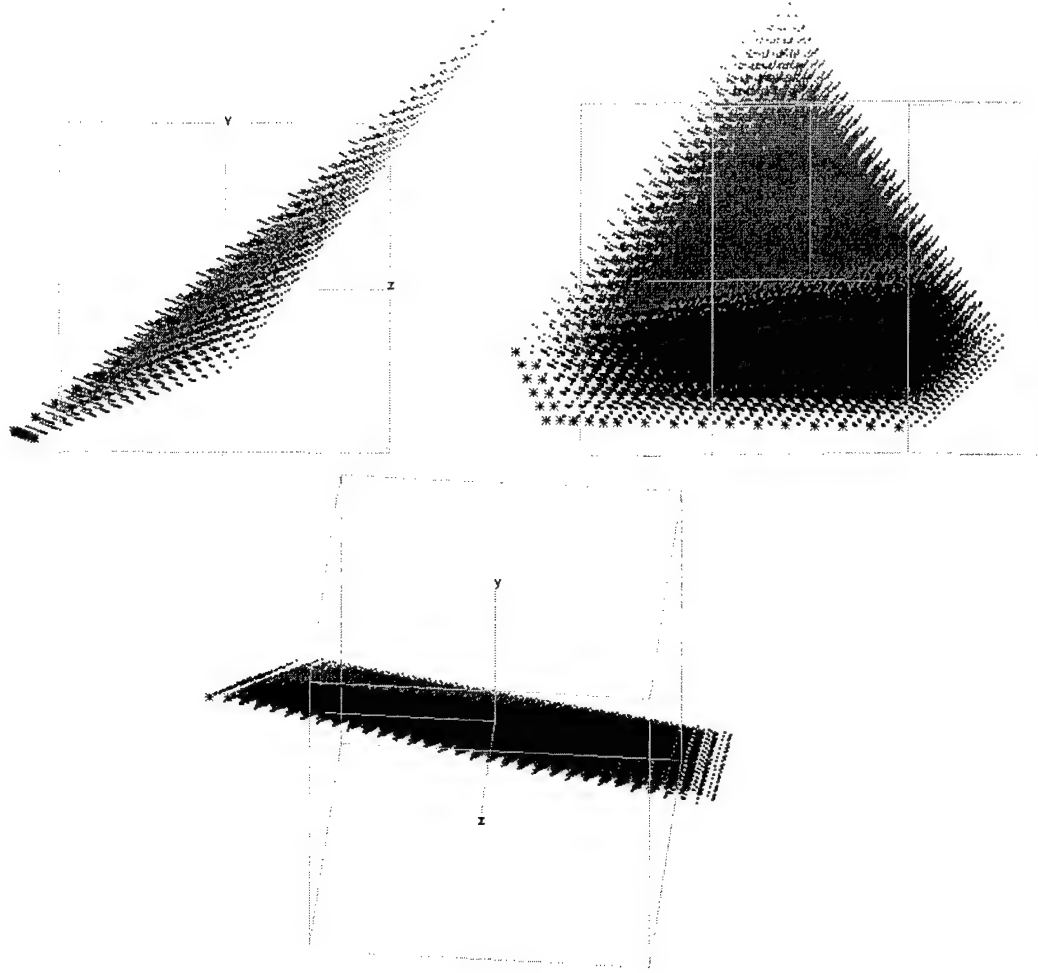


Figure 10. Three Views of the Target MOP Tri-objective Space for Resource Level 1

For Resource Level 1, the probability of selecting a feasible point is given by the quotient of the decision space cardinality and the binary value of the chromosome length (assuming a one-to-one mapping of the decision space to the objective space):

$$\frac{630,630}{2^{60}} \approx 5.47 \times 10^{-13} \quad (3.24)$$

Since the building blocks found by the algorithm are the result of converging to the infeasible front, the use of initially random templates was discarded in favor of user selected templates that are corrected to be feasible. The templates are defined prior the

start of the run, are different for each objective, and are not updated with the best individual after each era. Preliminary results indicate that this approach effectively stimulated the algorithm's search in feasible space.

Performance Measures

If PF_{true} is known, the *Final Generational Distance* metric can be used to characterize how “far” PF_{known} is from PF_{true} :

$$G \triangleq \frac{\left(\sum_{i=1}^n d_i^p \right)^{1/p}}{n} \quad (3.25)$$

where n is the number of vectors in PF_{known} , $p = 2$, and d_i is the Euclidean distance between each vector and the closest element of PF_{true} (Van Veldhuizen, 1999:6-15).

An MOEA adds elements to PF_{known} over a number of generations. The number of nondominated vectors that are added can vary depending on how much of the objective space that the algorithm is allowed to explore. The *Overall Nondominated Vector Generation* (ONVG) metric can be used to measure how “good” an MOEA is at generating desired solutions given a fixed fraction of search space to explore (Van Veldhuizen, 1999:6-18):

$$ONVG \triangleq |PF_{known}| \quad (3.26)$$

Alone, this metric says nothing about the quality of PF_{known} . Even if an MOEA's ONVG is equivalent to the number of vectors in PF_{true} , it may be that PF_{true} does not contain any element of PF_{known} . The *Error Ratio* metric takes this into consideration in order to characterize how well PF_{known} converges to PF_{true} :

$$E \triangleq \frac{\sum_{i=1}^n e_i}{n} \quad (3.27)$$

where n is the number of vectors in PF_{known} and

$$e_i = \begin{cases} 0 & \text{if vector } i, i = (1, \dots, n) \in PF_{true}, \\ 1 & \text{otherwise} \end{cases} \quad (3.28)$$

(Van Veldhuizen, 1999:6-14).

Experimental Design

Computational Environment. The MOMGA-II is written in *ANSI C* and compiled using the *Sun WorkShop Compiler version C 4.2*. It can be executed in any UNIX / LINUX environment. For this research, the execution platform is a Sun Ultra 10 workstation equipped with a 450 MHz processor, 1 GB RAM, and Solaris 2.8.

The target MOP is a discrete mapping from the integers to \mathbb{R}^3 . On a computer, real number representation is dependent on machine specific resolution. Therefore, determination of PF_{true} is machine dependent. However, for the three objectives—suitability, weight, and volume—accuracy on the order of 10^{-6} and beyond is not an issue.

Exhaustive deterministic enumeration is needed to find P_{true} and PF_{true} , limited by machine specific resolution, of course. A program that performs this task was written in *ANSI C* and compiled using *Microsoft Visual C++ 6.0*. The source code for this program is presented in Appendix C. The program code was not optimized and uses only a single list to maintain P_{known} and PF_{known} . The program was executed on a Dell Precision 610 equipped with a Pentium II 500MHz processor, 2GB RAM, and using *Windows 2000 Professional*. It was observed that this platform initially processes on average 40,000

solutions per minute but that this rate decreases exponentially with each additional solution. Using this program, complete enumeration of scenarios beyond index 1 in Table 4 would exceed the time requirement for this thesis by centuries. Therefore, absolute comparisons are only made between the enumerated P_{true} / PF_{true} and the MOMGA-II P_{known} / PF_{known} for index 1 in Table 4. The enumerated PF_{true} and P_{true} is in Appendix D.

Experiments. The first objective is to compare, in terms of absolute performance, the output of the MOMGA-II to the baseline exhaustive enumeration solution in order to *tune* the algorithm to the MOP formulation. There is no definitive answer as to whether this comparison should be made in decision or objective space. This decision is typically left to personal preference. For this research, objective space comparisons are preferred because of the high-dimensionality of the decision space. For statistical comparison purposes, all results will be taken from 30 replications of the MOMGA-II using a different random number generator seed each time. The *random()* function in the *random.c* file generates a single random number between 0.0 and 1.0 using the subtractive method described in (Knuth, 1981).

The second objective is to demonstrate the ability of the implicitly constraining MOMGA-II to produce level-wise nondominated force mix sets as defined in the section on model formulation. The third objective is to examine how execution time responds to target MOP problem size. These three objectives are met through the following experiments—absolute performance response to parameter changes, execution timing, and the demonstration of level-wise nondominated force mix sets.

Absolute Performance Response to Parameter Changes. The goal of this experiment is to identify how MOMGA-II effectiveness changes with different key parameter settings. There are no studies that show which parameters and what values are key to good performance for MOEAs (Van Veldhuizen, 1999:6-7). While a complete parameter analysis is warranted in this case, the allotted time for this research is limited and only certain primary operators and values can be investigated.

Experimental Parameters. The relationship between decision variables may affect algorithm performance. Different building block ranges will be used to take advantage of this possible relationship. In the MOMGA-II, search is primarily accomplished through building block filtering, splicing, and selection. The probability of cutting a string will be changed to investigate its affect on algorithm effectiveness. One of these settings will be a zero probability of cutting, placing the burden of search on building block filtering, splicing, and tournament selection. The splice operator used in the juxtapositional phase is the primary method of string composition in the MOMGA-II. Its probability will be reduced to reveal its affect on performance. Finally, initial population size affects the amount of fitness landscape information, and therefore building blocks, available to building block filtering. PF_{true} will be compared to the results of the MOMGA-II by individually applying each of the parameter settings listed in Table 6. ~~Table 6.~~ to the base settings listed in Table 7. A total of nine alternative parameter settings are used, each with 30 replications. The metrics G, ONVG, and E are used to quantitatively compare the alternatives.

Table 6. Experimental Parameter Settings

PARAMETER	SETTINGS
<i>BB size</i>	{ 2, 4, 8 }
p_{cut}	{ 0, 0.2, .2 }
p_{splice}	{ 1.0, 0.8, 0.6 }
<i>init. pop size</i>	{ 600, 915, 1200 }

Table 7. MOMGA-II Parameter Settings

PARAMETER	SETTING
p_{cut}	0.2
p_{splice}	1.0
p_m	0.0
t_{dom}	3
σ_{share}	Equation (2.18)
<i>eras</i>	4
Initial population size	915
termination	average string length \approx problem size

Execution Timing. This analysis does not take into account the additional run time needed to obtain an acceptable level of convergence and looks only at how execution timing is affected by increasing the problem size. For each index in Table 4, and using the base parameter settings in Table 7, a time hack will be taken from program start to the end of the juxtapositional phase in era 4. Thirty replications are used for each index to provide a good statistical sample.

Demonstration of Level-wise Nondominated Force Mix Sets. The goal of this experiment is to demonstrate the ability of the MOMGA-II to produce level-wise nondominated force mix sets in the absence of explicit constraint handling methods.

Since P_{true} and PF_{true} are unknown for scenarios beyond index 1, no statistical comparisons are made. Of particular interest is non-dominated set cardinality after applying the constraints.

MOMGA-II Parameter Settings. Unless otherwise stated, each replication of the MOMGA-II is performed using the settings listed in Table 7. Except for the termination rule, the listed parameter settings are the “default” settings used in previous research (Van Veldhuizen, 1999, Zydallis, et al. 2001b). The termination rule was selected to allow runs with larger building block sizes to be less dependent on template fitness.

Summary

This chapter addressed the second research question by presenting a model to describe the inputs and outputs to a MOP tool and formulating the research problem MOP in terms of its decision variables, objective functions, and constraints. The target MOP was constructed using inputs that provide linkage between this thesis and concurrent ALP research, and also create a search space of sufficient size with which to evaluate the MOP tool.

After discussing the selection of the MOMGA-II as the MOP tool, the experimental objectives, metrics, and key MOMGA-II parameter settings were presented in order to address the third research question: how to evaluate the selected MOP methodology. The chapter concluded with a discussion of the computational environment and the three experiments designed to meet experimental objectives. The next chapter addresses the fourth research question by analyzing MOMGA-II effectiveness and efficiency when applied to the target MOP.

IV. Results

Introduction

The previous chapter outlined the experimental methodology used to apply the MOMGA-II in three comparative experiments. This chapter focuses on the last research question, which addresses MOMGA-II solution quality and efficiency as it pertains to the research problem of optimizing MRR suitability and lift cost. The sections that follow present the analysis of the experimental data and report the results.

First, the results from parametric testing form the basis for an absolute comparison of the MOMGA-II to PF_{true} for a single resource level. Next, execution timing results are used comment on the time complexity of the MOMGA-II as it relates to problem scale. Finally, the results from level-wise runs are used to demonstrate the ability of the implicit constraint handling MOMGA-II to produce non-dominated sets of force mixes corresponding to various sortie resource levels.

Statistical Analysis

Absolute Performance Comparison. The sample data for Final Generational Distance, Figure 13 in Appendix F, suggests that the population distributions are not normal, requiring non-parametric methods for statistical comparisons. The technique employed, the Kruskal-Wallis H -test, assumes that the samples are random and independent, at least five measurements in each sample, and that the probability

distributions from which the samples are drawn are continuous (McClave, et al., 1998:892). These assumptions are satisfied and so the following hypotheses are tested:

H_0 : The probability distributions of the parameter groups for the G metric are the same.

H_A : At least two of the groups are different.

The H -test in Figure 13 of Appendix F was accomplished using JMPIN 4.0.2. At the 0.1 significance level, the observed significance level of 0.15 indicates that there is insufficient evidence to reject H_0 .

The ONVG sample data across the parameter groups are only borderline normally distributed and so the Kruskal-Wallis H -test is also applied here (Figure 14 in Appendix F). The ONVG sample data are cardinal and therefore Poisson distributed (Reynolds, 2001). For a large ONVG range (in this case from 0 to 630,630) the assumption of a continuous distribution is reasonable (Reynolds, 2001). The following hypotheses are tested:

H_0 : The probability distributions of the parameter groups for the $ONVG$ metric are the same.

H_A : At least two of the groups are different.

At the 0.1 significance level, the observed significance level of 0.29 indicates that there is insufficient evidence to reject H_0 .

These results show that using implicit constraint handling, MOMGA-II convergence properties and non-dominated set cardinality are not significantly affected by the choice of selected parameter values within the range tested. When infeasible solutions are not explicitly dealt with by the algorithm, it is relying primarily upon building block filtering and the juxtapositional generations to select and combine good

building blocks into feasible solutions. Target MOP pilot runs of the MOMGA-II without any modifications to the algorithm resulted in no feasible solutions at all. By simply specifying a feasible template for each objective and carrying them forward every era, non-dominated set cardinality improved to a mean of 5 with a standard deviation of 0.4.

Referring to Figures 15 – 23 in Appendix G, the three-dimensional plots of PF_{true} and PF_{known} for each parametric alternative allow qualitative assessments of MOMGA-II performance. Although the MOMGA-II never found any solutions in PF_{true} , the PF_{known} , nearly all of the non-dominated sets approximate the structure of PF_{true} and some of the solutions are very close to the front. That this occurred without the incorporation of any explicit constraint handling methods is indicative of the algorithm's robustness.

Execution Timing Analysis. This analysis does not take into account the additional run time needed to obtain acceptable level of convergence, looking only at how execution timing is affected by increasing the problem size. In all cases, the MOMGA-II completed 4 eras in under 25 seconds. It is seen from Figure 11 that within the range of 60 to 120 bits, execution time increases nearly linearly.

Without extrapolating, it is difficult to see how execution time responds to scaling up the test MOP to a real world size problem with seven basic aerospace missions and on the order of 100 or more MRR types. This preliminary result suggests that on the tested platform, MOMGA-II execution timing may scale linearly or possibly quadratically with chromosome length.

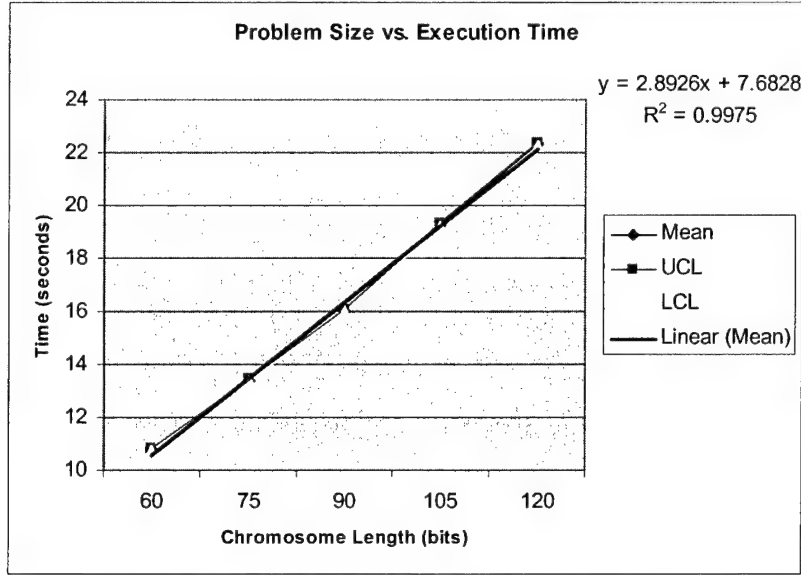


Figure 11. Problem Size vs. Execution Time

Demonstration of Level-wise Nondominated Force Mix Sets. The feasible solution set cardinalities for the two lowest resource levels are both six. The MOMGA-II found no nondominated feasible solutions at any of the other resource levels. The fractions of feasible points to total points corresponding to each resource level differ from each other by approximately an order of magnitude:

$$\begin{aligned}
 RL_1 &\approx \frac{6.30 \times 10^5}{2^{60}} \approx 5.47 \times 10^{-13} \\
 RL_2 &\approx \frac{1.60 \times 10^8}{2^{75}} \approx 4.22 \times 10^{-15} \\
 RL_3 &\approx \frac{2.56 \times 10^{12}}{2^{90}} \approx 2.07 \times 10^{-15} \\
 RL_4 &\approx \frac{1.48 \times 10^{16}}{2^{105}} \approx 3.65 \times 10^{-16} \\
 RL_5 &\approx \frac{4.37 \times 10^{19}}{2^{120}} \approx 3.29 \times 10^{-17}
 \end{aligned}$$

The fractions of feasible space are not grossly smaller at progressively larger resource levels and may not completely account for the algorithm's inability to find feasible, nondominated solutions above resource level two. With random population initialization, the algorithm may be misled right off the bat by starting with infeasible members. Nor does the use of feasible templates guarantee that feasible building blocks will be found during building block filtering. The same feasibility problems apply to the operations in the juxtapositional phase. Since the templates are not updated with the best found individual for each objective, this version of the MOMGA-II operates with a handicap. The overall result is that the algorithm is drawn to an infeasible front.

A recently updated version of the MOMGA-II incorporates a solution repair function that iteratively adjusts bits in a random manner until a solution becomes feasible, restricting the search to feasible space. The target problem was applied to the algorithm, but due to time constraints, proper tuning and complete analysis was not performed. Preliminary results are promising. Using the same base case parameters, random seeds, and 30 replications, the reported metrics in Table 8 are greatly improved.

Table 8. Explicit Constraint Metrics

	MEAN	STANDARD DEVIATION	MEDIAN
<i>G</i>	5.2	7.7	0.1
<i>ONVG</i>	22.7	1.5	23
<i>E</i>	49.8	7.7	47.9

When results are combined over 4 replications, the MOMGA-II finds all 26 points in PF_{true} for Resource Level 1.

When applied to Resource Levels 1 through 5, the MOMGA-II was able to produce feasible, equally preferred force mixes with the cardinalities listed in Table 9. Three-dimensional plots of PF_{known} for each level (Appendix F) suggest that a solution basic structure is maintained over all problem scales.

Table 9. Explicit Constraint MOMGA-II ONVG by Resource Level.

RESOURCE LEVEL	ONVG
1	26
2	70
3	131
4	215
5	301

Summary

This chapter answers research question four by presenting experimental analyses and results for three experiments that address the experimental objectives: absolute performance response to parameter changes, execution timing, and the demonstration of level-wise nondominated force mix sets. Overall, the results reveal the MOMGA-II's robustness and linear execution time (on the range tested), and show that implicit constraint handling as applied to the MOMGA-II does not go far enough to prevent convergence to an infeasible front. Preliminary results of an explicit constraint handling version of the MOMGA-II show greatly improved performance.

V. Conclusion

Introduction

Chapter I began with the motivation of this research, stating that the Defense Advanced research Projects Agency's Advanced Logistics Project (ALP) seeks to bring campaign planning into the 21st century with multiple, real-time deployment plans and rapid replanning. We can capitalize further on what ALP brings to the table by providing a methodology to evaluate multiple plans and provide the warfighter with the best available package with which to do the job.

As a front-end to ALP, the Mission-Resource Value Assessment Tool (M-R VAT) intends to assess competing force mixes in terms of their intrinsic task capability and the campaign specific issues affecting their effective employment in the theater. The primary goal of this research was to identify force mixes that, in terms of their intrinsic value, represent the best match of assets to tasks with the smallest deployment footprint.

To accomplish this goal, four research questions answered:

1. Which methodologies can be used to trade-off intrinsic value and deployment cost (lift) and result in a set of force mixes that are preferred over others?
2. What are the forms of the decision and objective spaces?
3. How is the selected approach evaluated?
4. Does the selected approach result in an acceptable solution to the research problem in a reasonable amount of time?

A series of research phases was used to answer these questions.

The first phase was a literature review intended to provide a broad overview of the multiobjective optimization problem (MOP) class in order to help with the selection of an appropriate methodology. The review highlighted concepts of MOP formulation, competing objectives, global versus local optimization, constrained optimization, and Pareto dominance of solutions. Also reviewed were classical and modern methodologies, including those from the metaheuristic class. The results of the literature review served to answer research question 1.

The next phase of this research incorporated the ALP Pilot Problem (Swartz, 1999) in the definition of the multiobjective model and formulation. The *Mission Ready Resource*, defined as a building block of capability, suitability, and deployment cost; the *Task Preference Vector*; and the definitions of acceptable force mixes defined the decision space, objective space, and the problem's constraints, thus answering research question 2. This phase also defined the target multiobjective problem used to evaluate the MOP model and formulation, and their application to the selected MOP methodology.

An important goal of this research was to incorporate as much problem domain knowledge as possible into the algorithmic approach. The third phase used the problem domain knowledge from the model and MOP formulation to select an appropriate MOP methodology. The Multiobjective Messy Genetic Algorithm (MOMGA-II), shown to robustly and flexibly handle a variety of difficult pedagogical problems, was selected to produce the desired nondominated sets of solution, thus answering research question 3. The algorithm was adapted to the level-wise output requirement of the MOP model. The MOMGA-II also incorporates implicit constraint handling in order to investigate that method's affect on the effectiveness of the algorithm.

Finally, the last phase evaluates the employed methodology based on three experimental objectives. The first objective was to compare, in terms of absolute performance, the output of the MOMGA-II to the baseline output of the exhaustive enumeration solution in order to help *tune* the algorithm to the MOP formulation. The second objective was to demonstrate the ability of the implicitly constraining MOMGA-II to produce level-wise nondominated force mix sets as defined by the model formulation. The third objective was to examine how execution time responds to target MOP problem size. The answer to research question 4 is found in the experimental results showing that implicit constraint handling as applied to the MOMGA-II is not a viable approach to producing acceptable sets of nondominated force mixes. However, the results also reveal the MOMGA-II's robustness and linear execution time (on the range tested). This, along with the greatly improved solution quality and cardinality achieved by preliminary runs of the solution repairing MOMGA-II support the viability of this approach to producing well balance force mixes.

Conclusions

This research shows that the multiobjective model and problem formulations, along the with test problem, are constructive approaches to investigating the problem of force mix selection. The importance of this research is in the illumination of problem complexity for so abstract a problem. Although simplified by assumptions, the employed methodology allows us to gauge problem complexity and uncover problem domain knowledge that must be incorporated into any solution platform. An important aspect of

problem complexity was revealed by mathematically defining the cardinality of the constrained multiobjective research problem decision space.

A program allowing for deterministic enumeration and Pareto dominance checking of a large solution space was developed to support the research methodology. In addition, a post-processing program was developed to analyze MOMGA-II output for solution dominance, feasibility, and uniqueness. Both programs were essential in supporting this methodology and have applications beyond it as part of a generic MOP tool kit.

While implicit constraint handling allows the MOMGA-II complete access to fitness landscape information, this research demonstrated that such a method is ineffective when the unconstrained Pareto front does not contain the constrained Pareto front. Despite this handicap, this research showed the adaptability of the MOMGA-II to the problem domain and viability of the platform for larger scale problems. Preliminary results indicate that, without parametric tuning, the solution repair function significantly contributes to algorithm convergence to feasible, optimal force mixes, warranting further investigation as an efficient and effective method for identifying well balanced force mixes.

Limitations

A number of simplifying assumptions were made for this research: single sortie, capability, linear suitability, and no limit on available lift or assets. These assumptions dull the real-world applicability of the devised model and must be dealt with before employing the model outside of the research realm.

The execution time experiment was conducted on a chromosome length range that, until now, had not been attempted. The reality of the situation, however, is that the number of USAF MRR types ranges in the hundreds and those of our allies compound this situation. While the “approximately linear” result of the experiment is positive, extrapolation of the algorithm’s time complexity to a real-world scaled problem is fraught with peril.

For practical reasons, parametric tests of the algorithm were limited in terms of range and number of parameters. This constitutes only a coarse tuning and is probably not the best combination of parameter values to use. As revealed by the literature review on genetic algorithms, parameter settings are typically problem dependent and little conclusive research on proper parameter settings exists. Parameter value selection is more art than science at this point.

A design consideration of the M-R VAT front-end is that it should run on a Microsoft Windows enabled PC so that it may be widely distributed throughout the planning community. At this time, the MOMGA-II code executes on Unix and Linux platforms. Significant effort is needed to port the code to run on a Microsoft platform.

As part of the methodology, problem domain knowledge was sought out and applied to the MOP tool *when found*. There may be elements of the problem that have escaped the eye of the researcher, elements which, when properly employed can effect the effectiveness and efficiency of the search algorithm.

The model used in this research is based strictly upon the intrinsic value of force mixes. The intrinsically scored output of the model is then extrinsically scored to produce a ranking of force mixes. However, force mixes that have composite intrinsic

and extrinsic values may have a different Pareto front, much less a different fitness landscape. Extrinsic knowledge constitutes problem domain knowledge and can be incorporated as part of the MOP formulation. Another tie-breaking methodology would need to be employed, but this too may also be incorporated as problem domain knowledge. The choice of where to stop with this reasoning may best depend on how much problem domain flexibility is lost or gained by the tool—a highly accurate assessment means nothing if the rules for assessment change and the tool is unable or too complex to adapt.

Recommendations

This research was limited in that only an implicit constraint handling method was employed in the MOMGA-II. Any complete assessment of the algorithm's utility to the research problem must include an explicit constraint handling version. Preliminary results from the MOMGA-II using a solution repair function showed greatly improved solution quality and cardinality, clearly indicating the direction of future efforts to identify well balanced force mixes.

When P_{true} or PF_{true} is unknown, some of the quantitative metrics employed by Van Veldhuizen can assess convergence to PF_{true} (1999:8-5). Relative quantitative performance reveals differences between alternatives but offers little in terms of solution quality. The best assessment is an absolute comparison of a MOP tool's solution output to P_{true} or PF_{true} . Complete enumeration of all possible solutions is limited, as it is an n^2 algorithm, n being the number of solutions in the space. It is recommended to completely enumerate at least three inflection points (sortie levels) on the Task Preference Vector in

order to quantitatively evaluate a selected algorithm's solution quality and move beyond empirical results. The IBM SP computers at both the Aeronautical Systems Center's Major Shared Resource Center (ASC MSRC) and the U.S. Army Corps of Engineers Waterways Experiment Station's (CEWES) MSRC can be used to deterministically enumerate all possible solutions for a given MOP at levels that are well beyond the computational capabilities of mere desktop machines.

Future Research

For the MOP formulated in this research, it is possible that the structure of PF_{true} is not sensitive to problem scale. The advantage in this case is that the search can be localized to the region of the objective space that PF_{true} lies for any number of decision variables and sortie level. As suggested in the recommendations section, it would be beneficial to completely enumerate several points on the Task Preference Vector to explore this possibility.

The research problem can also be applied to two or more different MOP solving approaches, all of which can be compared to the current deterministic approach in terms of flexibility, solution quality, ease of implementation, and scalability.

In terms of problem scale, chromosome length is a limiting factor to computational efficiency. Efficiency can be gained by employing variable length strings so that the length of the bit string representing a complete decision variable is independent of the others. This will reduce computer memory overhead and the size of the search space, but possibly at the expense of bias in favor of operating on longer length decision variables.

One way to handle the construction of force mix threads is through a post-processing algorithm. The algorithm operates on level-wise nondominated sets of force mixes and, starting at the lowest (or highest) resource level, constructs threads iteratively, taking into consideration all possible feasible threads. However, the worst case maximum number of threads to construct given K resource levels is

$$n_k \times n_{k+1} \times \cdots \times n_K = \prod_{k=1}^K n_k \quad (5.1)$$

where n is the number of MRRs to expend at a given resource level. Such a potentially large number of alternative force mixes would overwhelm a decision maker and must therefore be pared to a reasonable number. A way to do this that considers the best feasible threads is to use Filcek's extrinsic scoring model to rank the individual force mixes for each resource level (2001). Rather than randomly selecting a starting point and any following points, construction of feasible force mix threads proceeds using the highest ranked force mixes first. Not all of the possible feasible force mix threads need be constructed, but those that are constructed are based on the best intrinsically and extrinsically evaluated force mixes.

Finally, the simplifying assumptions used to initially explore the research problem may almost certainly have a large impact upon the problem domain and the fitness landscape. It is recommended that the need for these assumptions be evaluated and, where needed, supplanted by more realistic modeling information in order to enhance its operational applicability.

Summary

The multiobjective optimization problem model and formulation developed for the Mission-Resource Value Assessment Tool establish the fundamental form and complexity of force mix selection defined by the ALP Pilot Problem. The research methodology and its results are the first steps to providing rapid force mix selection based on task suitability, sortie capability, and the amount of finite lift resources consumed. Follow-up research is well-positioned to spring forward from this point and develop the multiobjective force mix assessment tool capable of rapidly providing best value / small footprint alternative force mixes corresponding to the desires of the warfighter.

Appendix A: Pareto Concepts

MOPs present a set of solutions from a trade-off surface between objectives. To be better understood, it is necessary to define key Pareto concepts: *Pareto Dominance*, *Pareto Optimality*, the *Pareto Optimal Set*, and the *Pareto Front*. Van Veldhuizen's definitions are noted for their consistency with theory and are the ones used in this thesis (1999:2-3):

Pareto Dominance: A vector $\vec{u} = (u_1, \dots, u_k)$ is said to *dominate* $\vec{v} = (v_1, \dots, v_k)$ (denoted by $\vec{u} \prec \vec{v}$) if and only if \vec{u} is partially less than \vec{v} , i.e., for all $i \in \{1, \dots, k\}$, $u_i \leq v_i$ and there exists an $i \in \{1, \dots, k\}$ such that $u_i < v_i$.

Pareto Optimality: A solution $x \in \Omega$ is said to be Pareto optimal with respect to Ω if and only if there is no $x' \in \Omega$ for which $\vec{v} = F(x') = (f_1(x'), \dots, f_k(x'))$ dominates $\vec{u} = F(x) = (f_1(x), \dots, f_k(x))$. The phrase "Pareto optimal" is taken to mean with respect to the entire decision variable space unless otherwise specified.

Pareto Optimal Set: For a given MOP $F(x)$, the Pareto optimal set (P^*) is defined as:

$$P^* := \{x \in \Omega \text{ such that there is no } x' \in \Omega \text{ where } F(x') \prec F(x)\} \quad (\text{A.1})$$

Pareto Front: For a given MOP $F(x)$ and Pareto optimal set P^* , the Pareto front (PF^*) is defined as:

$$PF^* := \{\vec{u} = F(x) = (f_1(x), \dots, f_k(x)) \mid x \in P^*\} \quad (\text{A.2})$$

P is the set of all solutions whose vectors are non-dominated with respect to all other vectors in the objective space. When mapped to the objective space, the solutions in P form the Pareto front, PF .

Appendix B: Advanced Logistics Program (ALP) Pilot Problem

A *Mission Ready Resource* (MRR) is a combination of an asset type and its resources, e.g. aircraft, pilot, fuel, munitions, support equipment and personnel, etc., that is designed to have a certain *suitability* for a single task. MRR suitability to a given task is measured per sortie on an absolute scale from 0 to 1, with 0 indicating no suitability and 1 indicating perfect suitability (Johnson, 2001). A combination of MRR types is defined to be a *MRR set* or *force mix*. To demonstrate, assume that a notional aircraft F , has two configurations, F_A and F_B , which constitutes two MRR types. Further, if the aircraft could be prepared and flown three times per day, then it would represent three MRRs per day. These three MRR's could either be all F_A configurations, or all F_B configurations, or some combination of the two configurations. Finally, assume three tasks: Suppression of Enemy Air Defenses (SEAD), Air-to-Air (AA), and Combat Air Support (CAS). Assuming negligible interaction between assets, the preference relationship between the assets and tasks is illustrated in Table 8.

Table 8. Asset-Mission Task Preference Matrix (A-M TPM) 1 (Swartz, 1999:1)

MRR	SEAD	AA	CAS
F_A	.3	1	.2
F_B	1	.1	.8

The critical physical dimensions of an asset are its *pallet weight* and *pallet volume*. A single MRR requires a given pallet weight and volume to support its deployment and use for a given sortie. The additional complexity of translating from

MRR space to asset space is beyond the scope of this research, hence the simplifying assumption that a MRR is equivalent to a single asset that is sampled without replacement, i.e. flies a single sortie per day.

We can consider a MRR as a building block of 1) task suitability, 2) pallet weight, and 3) pallet volume. Task suitability, pallet weight and pallet volume are hereafter referred to simply as *suitability*, *weight*, and *volume*, respectively.

A realistic assumption is that the decision maker's task preference may change depending on the number of resources at his / her disposal and the phase of the campaign. At the termination of a campaign, many if not all deployed resources are expected to redeploy to their origins or some other location. The ALP Pilot Problem is concerned strictly with the deployment planning and execution phase. So it also assumed that resource levels do not decrease, thus modeling a build-up of resources over time; and that the decision maker does not prefer fewer sorties over time. For instance, given a relatively low sortie generation level at the beginning of a campaign, a combatant commander whose goal is air dominance may prefer a ratio of 60 percent AA, 30 percent SEAD, and only 10 percent CAS. Over time, the sortie generation level increases and the next campaign phase may emphasize ground attack. This is reflected in the combatant commander's task preferences: 20 percent AA, 30 percent SEAD, and 50 percent CAS. The decision maker's task preference is explicitly indicated by the number of sorties, i.e. *capability*, desired for each task at a given resource level (sorties available). This leads naturally to a monotonic task preference vector with components that sum to the resource level. The points at which the ratio of desired capability changes are the inflection points. This is shown in Figure 12.

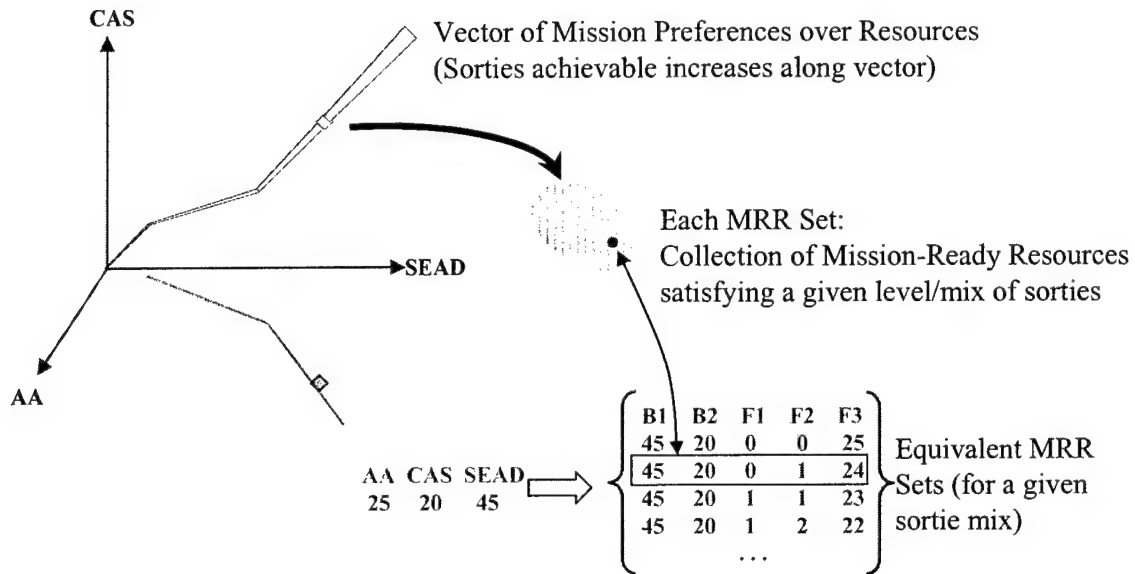


Figure 12. Matching Resources to Tasks (Johnson and Swartz, 2000b:16)

At the termination of a conflict, most if not all deployed resources are expected to redeploy to their origins or some other location. The ALP Pilot Problem is concerned strictly with the deployment planning phase. Two assumptions are made in the construction of the preference vector. First, it is assumed that resource levels do not decrease, thereby modeling a build-up of resources over time. Second, the preference vector is monotonic, i.e. the decision maker does not prefer fewer missions over time. We define a campaign *phase* as the time period where the commander's mission preferences remain constant. If the commander would decide that fewer missions were required—in a possibly different ratio of task types—then a new campaign phase would begin and the planning process described here would be repeated.

A decision maker can make his / her task capability preferences known at every feasible resource level, thereby minimizing interpolation error. A model with seven tasks

and a maximum resource level of 30 would require 210 task-resource allocation decisions, unreasonably overburdening the decision maker. Recognizing this, the Mission-Resource Value Assessment Tool (M-R VAT) model uses a reasonable number of inflection points to define the preference vector and interpolates preferences for intermediate points. The resulting interpolation error is assumed to result in a negligible difference between the preference vector and the decision maker's true task preferences.

The decision maker has a finite number of force mix choices with which to meet his / her task preferences. The number of force mixes for a single task is given by the following equation:

$$C_{n_k+m-1, n_k} = \frac{(n_k + m - 1)!}{n_k! (m - 1)!} \quad (\text{B.1})$$

where n_k is the desired sortie level for Task k , and m is the number of MRR types. For p tasks, the number of force mixes is

$$\prod_{k=1}^p C_{n_k+m-1, n_k} \quad (\text{B.2})$$

Using the example in Figure 12, a sortie mix of 25 AA, 20 CAS, and 45 SEAD results in approximately 5.35×10^{13} unique force mixes for the decision maker to consider. A look at the number of aircraft choices and configurations available in today's Air Force makes it clear that the number of MRR types ranges in the thousands and that the number of possible solutions grows unmanageably large.

Since an MRR is a building block of suitability, weight, and volume, the choice of the best MRR set depends on how it optimizes the three criteria: maximize suitability, minimize weight, and minimize volume. A MRR set is *acceptable* if, for a given

resource level, the decision maker is indifferent to the tradeoff between the three criteria of suitability, weight, and volume. Therefore, the formulation of acceptable force mixes is a multiobjective problem with competing objectives.

Appendix C: Source Code for ENUMERATION.C

```
/* ENUMERATION.C v2.4
 * Author: Dave Wakefield
 * email: wakester@earthlink.net
 * Date: 2/6/01
 * Reference: Author, "IDENTIFICATION OF PREFERRED OPERATIONAL PLAN
FORCE MIXES USING A
MULTIOBJECTIVE METHODOLOGY TO OPTIMIZE RESOURCE
SUITABILITY AND LIFT COST,"
Masters Thesis. Air Force Institute of Technology,
Wright-Patterson AFB, OH.
 * 2001.
 * This program performs a complete enumeration of a decision space
defined by 15 decision
 * variables. The output is a text file containing in column format the
nondominated
 * front for three objectives defined in the evaluate function, and the
corresponding
 * decision variables. Objective 1 is maximized, the others are
minimized. The program
 * takes as input a report filename and whether screen output of
progress is desired. A
 * progress file named 'progress.txt' is created and updated every 100th
solution.
 */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <conio.h>

double *Pcurrent; // Archive for Pareto front
int sizePcurrent; // Number of solutions in Pcurrent, not the
index; includes solutions flagged for removal
double fitness[3] = { 0, 0, 0 }; // initialize fitness vector

void evaluate( double f[], int i1, int i2, int i3, int i4, int i5, int
i6, int i7, int i8, int i9, int i10, int i11, int i12, int i13, int i14,
int i15 );
int pareto( int n );
double factorial( double a );
double mod( double a, int b );
void clear_kb( void );

main ()
{
    int Task1; // DM task 1 preference
    int Task2; // DM task 2 preference
    int Task3; // DM task 3 preference
    int maxX11; // Upper bound for X11 (task 1, MRR type 1)
    int maxX12; // Upper bound for X12 (task 1, MRR type 2)
```

```

int      maxX13; // Upper bound for X13 (task 1, MRR type 3)
int      maxX14; // Upper bound for X14 (task 1, MRR type 4)
int      maxX15; // Upper bound for X15 (task 1, MRR type 5)
int      maxX21; // Upper bound for X21 (task 2, MRR type 1)
int      maxX22; // Upper bound for X22 (task 2, MRR type 2)
int      maxX23; // Upper bound for X23 (task 2, MRR type 3)
int      maxX24; // Upper bound for X24 (task 2, MRR type 4)
int      maxX25; // Upper bound for X25 (task 2, MRR type 5)
int      maxX31; // Upper bound for X31 (task 3, MRR type 1)
int      maxX32; // Upper bound for X32 (task 3, MRR type 2)
int      maxX33; // Upper bound for X33 (task 3, MRR type 3)
int      maxX34; // Upper bound for X34 (task 3, MRR type 4)
int      maxX35; // Upper bound for X35 (task 3, MRR type 5)
int      numTasks = 3; // Number of tasks
int      numMRRs = 5; // Number of MRR types
int      i; // used to print report
int      i1, i2, i3, i4, i5, i6, i7, i8, i9, i10, i11, i12,
i13, i14, i15; // used in for statements
int      n; // Counts solutions for Pareto dominance testing
int      z; // expression for switch statement
int      nondominated; // 1 if solution is nondominated, 0
otherwise
int      t; // progress report solution increment
time_t   start, finish, time2t; // start, finish, elapsed time
// double aa, bb, cc;
double   duration; // program execution time
double   cardDS; // Calculated cardinality of the decision
space
double   countDS = 0; // Counter for number of solutions
evaluated
double   progress; // percentage of cardDS evaluated
FILE     *fp1;
FILE     *fp2;
char     filename[20];
char     *progressfile = "progress.txt";
char     output, retry;

/* Print program description to screen */
printf("ENUMERATION.C v2.4\nAuthor: Dave Wakefield\nemail:
wakester@earthlink.net\n");
printf("Date: 2/6/01\nReference: Author, IDENTIFICATION OF
PREFERRED OPERATIONAL PLAN");
printf("FORCE MIXES USINGA MULTIOBJECTIVE METHODOLOGY TO OPTIMIZE
RESOURCE SUITABILITY ");
printf("AND LIFT COST, Masters Thesis. Air Force Institute of
Technology, ");
printf("Wright-Patterson AFB, OH. 2001.\n\nThis program performs
a complete enumeration of ");
printf("a decision space defined by 15\ndecision variables. The
output is a text file ");
printf("containing in column format the\nnondominated front for
three objectives defined in ");
printf("the evaluate function, and\nthe corresponding decision
variables. Objective 1 is maximized, the others are minimized. The ");

```

```

    printf("program takes as input user-defined sortie levels for
Tasks 1\nthrough 3, a report filename, and whether screen output of
progress is ");
    printf(" desired.A progress file named 'progress.txt' is created
and updated every 100th\nsolution.\n\n");

    /* Get user defined task levels. Allows user reinput levels as
desired. */
    while ( 1 ) {
        puts( "\nEnter number of sorties desired for Task 1" );
        scanf( "%d", &Task1 );
        clear_kb();
        puts( "\nEnter number of sorties desired for Task 2" );
        scanf( "%d", &Task2 );
        clear_kb();
        puts( "\nEnter number of sorties desired for Task 3" );
        scanf( "%d", &Task3 );
        clear_kb();

        /* Calculate decision space cardinality using equation 3.2
in thesis */
        cardDS = factorial( (double)( Task1 + numMRRs - 1 ) ) / (
factorial( (double)( Task1 ) ) * factorial( (double)( numMRRs - 1 ) ) )
        * factorial( (double)( Task2 + numMRRs - 1 ) ) /
( factorial( (double)( Task2 ) ) * factorial( (double)( numMRRs - 1 ) ) )
        * factorial( (double)( Task3 + numMRRs - 1 ) ) /
( factorial( (double)( Task3 ) ) * factorial( (double)( numMRRs - 1 )
)));

        printf( "\nCalculated decision space cardinality = %f\n\n",
cardDS );
        printf( "Press 'r' to retry, or press Enter to continue.\n"
);
        retry = getch();
        if ( retry != 'r' ) {
            break;
        } // end if
    } // end while

    clear_kb();

    puts("Enter filename.");
    gets( filename );
    puts("\nEnter progress report increment, e.g. 100 for every 100th
solution.\n");
    scanf( "%d", &t );
    puts("\nEnter 's' for screen output of progress, or press Enter to
continue.\n");
    output = getch();

    /* Open progress file */
    if ( ( fp2 = fopen( progressfile, "w" ) ) == NULL ) {
        fprintf( stderr, "Error opening progress file." );
        exit(1);
    } // end if

```



```

/* Write header to progress file */
fprintf( fp2, "Solution number\tElapsed time (seconds)\n" );
fclose( fp2 );

/* allocate memory for 1st nondominated solution (3 fitness values
+ 15 DVs) */
Pcurrent = (double *) realloc(NULL, (18 * sizeof(double)));

/* memory allocation test */
if ( Pcurrent == NULL ) {
    puts("Memory allocation error.");
    exit(1);
} // end if

/* initialize 1st solution with negatives if maximizing */
Pcurrent[0] = -1; // max
for ( i = 1; i <= 17; i++ ) {
    Pcurrent[i] = 100000000; // min
} // end for
sizePcurrent = 1;

maxX11 = Task1;
maxX12 = Task1;
maxX13 = Task1;
maxX14 = Task1;
maxX15 = Task1;
maxX21 = Task2;
maxX22 = Task2;
maxX23 = Task2;
maxX24 = Task2;
maxX25 = Task2;
maxX31 = Task3;
maxX32 = Task3;
maxX33 = Task3;
maxX34 = Task3;
maxX35 = Task3;

/* Record start time */
start = time(0);

/* increment 1st DV */
for (i1 = 0; i1 <= maxX11; i1++) {

    /* increment 2nd DV */
    for (i2 = 0; i2 <= maxX12; i2++) {

        /* increment 3rd DV */
        for (i3 = 0; i3 <= maxX13; i3++) {

            /* increment 4th DV */
            for (i4 = 0; i4 <= maxX14; i4++) {

                /* increment 5th DV */
                for (i5 = 0; i5 <= maxX15; i5++) {

```

```

/* increment 6th DV */
for (i6 = 0; i6 <= maxX21; i6++) {

/* increment 7th DV */
for (i7 = 0; i7 <= maxX22; i7++) {

/* increment 8th DV */
for (i8 = 0; i8 <= maxX23; i8++) {

/* increment 9th DV */
for (i9 = 0; i9 <= maxX24; i9++) {

/* increment 10th DV */
for (i10 = 0; i10 <= maxX25; i10++) {

/* increment 11th DV */
for (i11 = 0; i11 <= maxX31; i11++) {

/* increment 12th DV */
for (i12 = 0; i12 <= maxX32; i12++) {

/* increment 13th DV */
for (i13 = 0; i13 <= maxX33; i13++) {

/* increment 14th DV */
for (i14 = 0; i14 <= maxX34; i14++) {

/* increment 15th DV */
for (i15 = 0; i15 <= maxX35; i15++) {

/* test that sum of like-task DVs is exactly the DV's
task preference */
if ( i1 + i2 + i3 + i4 + i5 == Task1
    && i6 + i7 + i8 + i9 + i10 == Task2
    && i11 + i12 + i13 + i14 + i15 == Task3) {

/* evaluate objective functions */
evaluate( fitness, i1, i2, i3, i4, i5, i6, i7,
i8, i9, i10, i11, i12, i13, i14, i15 );

/* initialize counter */
n = 0;

/* initialize nondomination flag; it's only
changed if new solution is dominated */
nondominated = 1;

/* while n < the number of solutions in Pcurrent
*/
while ( n < sizePcurrent ) {
    z = pareto( n );
    switch ( z ) {

```

```

/* Solution n in Pcurrent is dominated by
new solution & flagged for removal by setting the 1st
fitness value to a negative number
*/
case 1: {
    Pcurrent[18 * n] = -1;
    break;
} // end case 1

/* new solution is dominated, stop Pareto
testing, set domination flag */
case 2: {
    n = (sizePcurrent);
    nondominated = 0;
    break;
} // end case 2
/* new solution is indifferent and will be
added to Pcurrent */
default: {
    break;
} // end default
} // end switch
n++;
} // end while

/* increment count of solutions evaluated */
countDS++;

/* Diagnostic line */
printf( "%f solutions evaluated\n", countDS );

/* print progress every t solutions evaluated */
if ( mod( countDS,t ) == 0 ) {

    /* record time every t solutions */
    time2t = time(0);

    /* Calculate time to process 100 solutions
*/
    duration = difftime( time2t, start );

    /* Calculate percent complete of all
solutions */
    progress = 100 * ( countDS / cardDS );

    /* Reopen progress file for appending */
    if ( ( fp2 = fopen( progressfile, "a" ) )
== NULL ) {
        fprintf( stderr, "Error opening
progress file." );
        exit(1);
    } // end if

    /* Write progress to progress file */

```

```

duration );

fprintf( fp2, "%f\t%f\n", countDS,
fclose( fp2 );

/* Used when screen output of progress is
desired */
if ( output == 's' ) {
    printf("Time to solution %f = %f
seconds.\n", countDS, duration );
    printf( "%f percent completed\n\n",
progress );
} // end if
} // end if

/* if new solution is nondominated, add new
solution to the end of Pcurrent */
if ( nondominated == 1 ) {

    /* here's some more memory; increases by 3
doubles worth of bytes */
    Pcurrent = (double *) realloc(Pcurrent,
(18 * sizePcurrent * sizeof(double) + 18 * sizeof(double)));

    /* memory allocation test */
    if ( Pcurrent == NULL ) {
        puts("Memory allocation error.");
        exit(1);
    } // end if

    /* set values for new memory that was
allocated */
    Pcurrent[18 * sizePcurrent] = fitness[0];
    Pcurrent[18 * sizePcurrent + 1] =
fitness[1];
    Pcurrent[18 * sizePcurrent + 2] =
fitness[2];
    Pcurrent[18 * sizePcurrent + 3] = i1;
    Pcurrent[18 * sizePcurrent + 4] = i2;
    Pcurrent[18 * sizePcurrent + 5] = i3;
    Pcurrent[18 * sizePcurrent + 6] = i4;
    Pcurrent[18 * sizePcurrent + 7] = i5;
    Pcurrent[18 * sizePcurrent + 8] = i6;
    Pcurrent[18 * sizePcurrent + 9] = i7;
    Pcurrent[18 * sizePcurrent + 10] = i8;
    Pcurrent[18 * sizePcurrent + 11] = i9;
    Pcurrent[18 * sizePcurrent + 12] = i10;
    Pcurrent[18 * sizePcurrent + 13] = i11;
    Pcurrent[18 * sizePcurrent + 14] = i12;
    Pcurrent[18 * sizePcurrent + 15] = i13;
    Pcurrent[18 * sizePcurrent + 16] = i14;
    Pcurrent[18 * sizePcurrent + 17] = i15;

    /* increment count of solutions in
Pcurrent */
    sizePcurrent++;

```



```

        if ( Pcurrent[18*n] > fitness[0] && Pcurrent[18*n + 1] ==
fitness[1] && Pcurrent[18*n + 2] == fitness[2]
        || Pcurrent[18*n] == fitness[0] && Pcurrent[18*n + 1] ==
fitness[1] && Pcurrent[18*n + 2] < fitness[2]
        || Pcurrent[18*n] == fitness[0] && Pcurrent[18*n + 1] <
fitness[1] && Pcurrent[18*n + 2] == fitness[2]
        || Pcurrent[18*n] == fitness[0] && Pcurrent[18*n + 1] <
fitness[1] && Pcurrent[18*n + 2] < fitness[2]
        || Pcurrent[18*n] > fitness[0] && Pcurrent[18*n + 1] ==
fitness[1] && Pcurrent[18*n + 2] < fitness[2]
        || Pcurrent[18*n] > fitness[0] && Pcurrent[18*n + 1] <
fitness[1] && Pcurrent[18*n + 2] == fitness[2]
        || Pcurrent[18*n] > fitness[0] && Pcurrent[18*n + 1] <
fitness[1] && Pcurrent[18*n + 2] < fitness[2] ) {
            /* new solution is dominated by solution n in Pcurrent */
            m = 2;
            return m;
        } // end if

        m = 3;
        return m;
    }

double factorial( double a )
{
    if ( a == 1 )
        return 1;
    else {
        a *= factorial( a - 1 );
        return a;
    }
}

double mod( double a, int b )
{
    double c;

    /* using ceil function since I can't find a rounding funtion */
    c = ceil( fmod( a, (double)b ) * b );
    return c;
}

void clear_kb( void )
{
    /* Clears stdin of any waiting characters. */
    char junk[80];
    gets( junk );
}

```

Appendix D: P_{true} and PF_{true} for Table 4, Index 1

	OBJECTIVE 1	OBJECTIVE 2	OBJECTIVE 3	x11	x12	x13	x14	x15	x21	x22	x23	x24	x25	x31	x32	x33	x34	x35
0.815	318.4	27280	0	0	0	10	0	0	0	0	0	5	0	0	0	0	1	0
1.614	318.7	27225	1	0	0	9	0	0	0	0	0	5	0	0	0	0	1	0
2.413	319	27170	2	0	0	8	0	0	0	0	0	5	0	0	0	0	1	0
3.212	319.3	27115	3	0	0	7	0	0	0	0	0	5	0	0	0	0	1	0
4.011	319.6	27060	4	0	0	6	0	0	0	0	0	5	0	0	0	0	1	0
4.81	319.9	27005	5	0	0	5	0	0	0	0	0	5	0	0	0	0	1	0
5.609	320.2	26950	6	0	0	4	0	0	0	0	0	5	0	0	0	0	1	0
6.408	320.5	26895	7	0	0	3	0	0	0	0	0	5	0	0	0	0	1	0
7.207	320.8	26840	8	0	0	2	0	0	0	0	0	5	0	0	0	0	1	0
8.006	321.1	26785	9	0	0	1	0	0	0	0	0	5	0	0	0	0	1	0
8.805	321.4	26730	10	0	0	0	0	0	0	0	0	5	0	0	0	0	1	0
12.001	355.8	29810	10	0	0	0	0	0	0	4	0	1	0	0	0	0	1	0
12.8	364.4	30580	10	0	0	0	0	0	0	5	0	0	0	0	0	0	1	0
9.204	321.7	26675	10	0	0	0	0	0	1	0	0	4	0	0	0	0	1	0
11.601	347.5	28985	10	0	0	0	0	0	1	3	0	1	0	0	0	0	1	0
12.4	356.1	29755	10	0	0	0	0	0	1	4	0	0	0	0	0	0	1	0
9.603	322	26620	10	0	0	0	0	0	2	0	0	3	0	0	0	0	1	0
11.201	339.2	28160	10	0	0	0	0	0	2	2	0	1	0	0	0	0	1	0
12	347.8	28930	10	0	0	0	0	0	2	3	0	0	0	0	0	0	1	0
10.002	322.3	26565	10	0	0	0	0	0	3	0	0	2	0	0	0	0	1	0
10.801	330.9	27335	10	0	0	0	0	0	3	1	0	1	0	0	0	0	1	0
11.6	339.5	28105	10	0	0	0	0	0	3	2	0	0	0	0	0	0	1	0
10.401	322.6	26510	10	0	0	0	0	0	4	0	0	1	0	0	0	0	1	0
11.2	331.2	27280	10	0	0	0	0	0	4	1	0	0	0	0	0	0	1	0
10.8	322.9	26455	10	0	0	0	0	0	5	0	0	0	0	0	0	0	1	0
10.001	323.2	26400	10	0	0	0	0	0	5	0	0	0	0	1	0	0	0	0

Appendix E: Source Code for Pareto_processing.c

```
/*=====
Pareto_processing.c V2.2
2/18/01
Dave Wakefield & Jesse Zydallis. Thanks to Matt Johnson.
Side constraint, Pareto check, & clone check program.
This program will take the data points and remove those that are
1) infeasible, 2) dominated, and 3) clones. These three routines are
run successively on an input file, generating a report after each
routine. However, input and output file names are hard coded, along
with defined parameters. The data array size in the main function is
also hard coded.
=====*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define NUM_DVS 15;
#define NUM_FUNCS 3;
#define MAX_COLS 18;
#define MAX_PTS 40000; /* set very high since number of solutions is
unknown */
#define PAGEWIDTH 63;
#define REPS 30;
#define MAXIMIZATION_FLAG 0; /* 1 means max */

int num_DVs, num_funcs;
num_DVs = NUM_DVS;
num_funcs = NUM_FUNCS;

int constraint( double *ex );
void is_par( char *filename, int rows, int file_num );
void is_clone( char *filename, int rows, int file_num );

main() {

    register int i, j, m; /* for loop indices */
    int pass; /* represents boolean result from constraint test */
    int nind, column; /* counts input file rows and columns */
    int total; /* used to detect end of input file row condition */
    int z; /* counts number of feasible solutions */
    int reps, max_pts;
    double *ex; /* array holding DVs for single solution */
    double number; /* temp var used to read in data from input file
*/
    FILE *fp, *fp2; /* fp reads, fp2 writes */
    char file1[30], file2[30]; /* file1 and file2 names */
    double data[40000][18]; /* array to hold input file data */

    total = MAX_COLS;
```

```

    reps = REPS;
    max_pts = MAX_PTS;

    /* outer loop used to process input files from all replications
*/
    for ( m = 1; m <= reps; m++ ) {
        /* set file1 to desired input file name and open it for
reading */
        if ( m < 10 ) {
            sprintf( file1, "e1_r0%d.pts", m );
        } else {
            sprintf( file1, "e1_r%d.pts", m );
        } /* end if */
        if ( ( fp = fopen( file1, "r" ) ) == NULL ) {
            fprintf( stderr, "Error opening file %s.", file1 );
            exit(1);
        } /* end if */

        column = nind = z = 0;

        /* read in data from input file */
        while( ( fscanf( fp, "%lf", &number ) ) != EOF ) {
            data[nind][column] = number;
            column++;
            column%=total; /* detects end of row */
            if ( column==0 ) { /* if end of row, move to next row
*/
                nind++;

                if ( nind==max_pts ) {
                    printf("Too many data points for
side_constraint.\n");
                    printf("Increase 'MAX_PTS'\n");
                    exit(1);
                } /* end if */
            } /* end if */
        } /* end while */
        fclose(fp);

        /* allocate memory to hold a row of DVs and OFs */
        if( !( ex = (double *)malloc( total*sizeof(double) ) ) ) {
            fprintf( stderr, "Insufficient memory for variable,
ex.\n");
        } /* end if */

        /* set file2 to desired out file name and open it for
writing */
        if ( m < 10 ) {
            sprintf( file2, "e1_r0%d.feas", m );
        } else {
            sprintf( file2, "e1_r%d.feas", m );
        } /* end if */
        if ( ( fp2 = fopen( file2, "w" ) ) == NULL ) {
            fprintf( stderr, "Error opening file %s.", file2 );
            exit(1);
        }
    }

```

```

        } /* end if */

        /* this loop checks each row of data array for feasibility
*/
        for ( i = 0; i < nind; i++ ) {

            for ( j = 0; j < num_DVs; j++ ) {
                *(ex+j) = data[i][j];
            } /* end for */

            pass = constraint(ex);
            if ( pass == 1 ) {
                for ( j=0; j< total; j++ ) {
                    fprintf( fp2, "%3.14f ", data[i][j] );
                } /* end for */
                fprintf( fp2, "\n" );
                z++;
            } /* end if */
        } /* end for */
        fclose( fp2 );

        /* pass feasible points file to Pareto check */
        is_par( file2, z, m );

    } /* end for */
} /* end main */

int constraint(double *ex)
{
    /* printf("ex = %f\n",*ex); */
    /*insert function here*/
    if ( ((*ex + *(ex+1) + *(ex+2) + *(ex+3) + *(ex+4)) == 10.0) &&
        ((*ex+5) + *(ex+6) + *(ex+7) + *(ex+8) + *(ex+9)) == 5.0)
    &&
        ((*ex+10) + *(ex+11) + *(ex+12) + *(ex+13) + *(ex+14)) ==
1.0)
    ) {
        return (1);
    } else {
        return (0);
    }
}

/* if ( ( *ex*(ex+1) + *(ex+1)*(ex+1))<=225) && ( ( *ex-3*(ex+1))
)<=-10 ) )*/
}

/*=====
function : is_par

purpose : finds pareto optimal points

developed : 2001 from newis_par.c by Matt Johnson

modified by Dave Wakefield & Jesse Zydallis
=====*/
void is_par( char *filename, int rows, int file_num )

```

```

{
    register int i, j, k;
    int flag1, flag2, count, rank, total, maximizationflag;
    FILE *fp, *fp2;
    double number, *answer_data, *answer_ptr;
    double *data, *ptr1, *ptr2, *ptr3;
    double frac_par;
    char file1[30];

    maximizationflag = MAXIMIZATION_FLAG;
    total = num_DVs + num_funcs;

    printf("Inside Pareto Analysis Routine!!!\n");

    data = (double *)malloc(rows*total*sizeof(double) ) ;
    if (data==NULL) {
        fprintf( stderr, "Not enough memory for output data.\n");
    } /* end if */

    answer_data = (double *)malloc(rows*total*sizeof(double) ) ;
    if (answer_data==NULL) {
        fprintf( stderr, "Not enough memory for output data.\n");
    } /* end if */

    /* open feasible data file, *.feas */
    if (( fp = fopen( filename, "r")) == NULL ) {
        fprintf( stderr, "Error opening file %s.", filename );
        exit(1);
    } /* end if */

    ptr1 = data;

    while((fscanf(fp, "%lf", &number))!=EOF){
        *ptr1=number;          /*input all the data*/
        ptr1++;
    } /* end while */

    fclose(fp);

    count=0;                  /*will be the number of pareto optimal
pts*/

    if (maximizationflag==0) { /*a minimization problem*/
        for (i=0; i<rows; i++) {
            ptr1=data+i*total;      /*beginning of each row*/
            rank=rows;
            ptr2=ptr1+num_DVs;      /*start at f1 for each row*/
            for (j=0; j<rows; j++) {
                ptr3=data+j*total+num_DVs; /*go through all
rows*/
                flag1=flag2=0;
                for (k=0; k < num_funcs; k++) {          /*go
through all the functions of a row*/
                    if ((*ptr2+k)<(*ptr3+k)) {

```

```

                                flag1++;    /*row can't be
dominated so break*/
                                break;
                                } /* end if */
                                if ((*ptr2+k)==(*ptr3+k)) {
                                    flag2++;
                                } /* end if */
                                } /* end for */
                                if (flag1>0 || flag2 == num_funcs) { /*non-
dominated || same row*/
                                    rank--;
                                } else {
                                    break;    /*a row is dominated,
move on*/
                                } /* end if */
                                } /* end for */
                                if (rank==0) {
                                    answer_ptr=answer_data+count*total; /*move down
count rows*/
                                    for(j=0; j<total; j++) {
                                        *(answer_ptr+j)=*(ptr1+j);    /*take
points and function values*/
                                    } /* end for */
                                    count++;
                                } /* end if */
                                } /* end for */
                                } else { /*a maximization problem*/
                                    for (i=0; i<rows; i++)
                                    {
                                        ptr1=data+i*total;    /*beginning of each row*/
                                        rank=rows;
                                        ptr2 = ptr1 + num_DVs; /* start at f1 for each row
*/
                                        for (j=0; j<rows; j++)
                                        {
                                            ptr3=data+j*total + num_DVs; /*go through all
rows*/
                                            flag1=flag2=0;
                                            for (k=0;k < num_funcs; k++) /*go through all
the functions of a row*/
                                            {
                                                if ((*ptr2+k)>(*ptr3+k))
                                                {
                                                    flag1++;    /*row can't be
dominated so break*/
                                                    break;
                                                }
                                                if ((*ptr2+k)==(*ptr3+k))
                                                {
                                                    flag2++;
                                                }
                                            }
                                            if (flag1>0 || flag2 == num_funcs )/*non-
dominated || same row*/
                                            {

```

```

                                rank--;
                                }
                                else
                                {
                                    break;                /*a row is dominated,
move on*/
                                }
                                }
                                if (rank==0)
                                {
                                    answer_ptr=answer_data+count*total; /*move down
count rows*/
                                    for(j=0; j<total; j++)
                                    {
                                        *(answer_ptr+j)=*(ptr1+j);    /*take
points and function values*/
                                    }
                                    count++;
                                }
                            }
                        }
                        free(data);

                        if ( file_num < 10 ) {
                            sprintf( file1, "e1_r0%d.prto", file_num );
                        } else {
                            sprintf( file1, "e1_r%d.prto", file_num );
                        } /* end if */
                        if (( fp2 = fopen( file1, "w")) == NULL ) {
                            fprintf( stderr, "Error opening file %s.", file1 );
                            exit(1);
                        } /* end if */
                        for (i=0; i<(total*count); i++)          { /* write the data to the
*.prto file */
                            fprintf(fp2, "%3.14f ", *(answer_data+i));
                            if ( (i+1)%total==0 ) { /*would mean finished a whole row*/
                                fprintf(fp2, "\n");
                            } /* end if */
                        } /* end for */
                        fclose(fp2);

                        free(answer_data);          /*free memory*/

                        frac_par=(double)count/(double)rows;
                        printf("%s fraction of pareto optimal points = %3.14f\n", file1,
frac_par);

                        /* pass Pareto points to is_clone to remove duplicate points */
                        is_clone( file1, count, file_num );
                    } /* end is_par */

/*=====
function : is_clone

purpose : finds duplicate points

```

developed : 2001 from newis_par.c by Matt Johnson

modified by Dave Wakefield & Jesse Zydallis

```
=====*/
void is_clone( char *filename, int rows, int file_num ) {

    register int i, j, k;
    int total; /* number of elements in a row */
    int count; /* counts the number of unique pareto optimal pts */
    FILE *fp, *fp2;
    double number, *answer_data, *answer_ptr;
    double *data, *ptr1, *ptr2;
    char file1[30];

    total = num_DVs + num_funcs;

    printf( "Inside No Clone Routine!!!\n" );

    data = (double *)malloc( rows * total * sizeof(double) ) ;
    if ( data == NULL ) {
        fprintf( stderr, "Not enough memory for output data.\n" );
    } /* end if */

    answer_data = (double *)malloc( rows * total * sizeof(double) ) ;
    if ( answer_data == NULL ) {
        fprintf( stderr, "Not enough memory for output data.\n" );
    } /* end if */

    /* open Pareto points data file, *.prto */
    if (( fp = fopen( filename, "r" )) == NULL ) {
        fprintf( stderr, "Error opening file %s.", filename );
        exit(1);
    } /* end if */

    ptr1 = data;

    while( ( fscanf( fp, "%lf", &number) ) != EOF ) {
        *ptr1 = number; /* input all the data */
        ptr1++;
    } /* end while */

    fclose( fp );

    count = 0;

    /* next 3 for loops compare COLUMN k of ROW i against COLUMN k of
ROW j */
    for ( i = 0; i < rows; i++ ) {
        ptr1 = data + i * total; /* point to beginning of ROW i */
        if ( *ptr1 != -1 ) { /* -1 means the row is a clone and it
won't be tested */
            for ( j = 0; j < rows; j++ ) {
                ptr2 = data + j * total; /* point to beginning
of ROW j */

```

```

        if ( i != j && *ptr2 != -1 ) { /* don't want to
compare a row against itself; don't want a duplicate row */
            for ( k = 0; k < num_DVs; k++ ) {
                if ( ( *(ptr1 + k) ) != ( *(ptr2 +
k) ) ) { /* point to COLUMN k */
                    /* ROW i is not a duplicate of ROW
j, so break to next j */
                    break;
                } /* end if */

                if ( k == ( num_DVs - 1 ) ) { /* if
last DV has been checked, ROW j is a clone of ROW i */
                    data[j * total] = -1; /* set
ROW j clone flag */
                } /* end if */
            } /* end for */
        } /* end if */
    } /* end for */

    /* if you get this far, then ROW i, having flagged
any clones of itself, is unique, so copy it to answer_data */
    answer_ptr = answer_data + count * total; /* move
down <count> rows to append ROW i to answer_data */
    for ( j = 0; j < total; j++ ) {
        *(answer_ptr+j) = *(ptr1+j); /* copy DVs
and function values */
    } /* end for */

    count++;
} /* end if */
} /* end for */

free( data );

if ( file_num < 10 ) {
    sprintf( file1, "e1_r0%d.front", file_num );
} else {
    sprintf( file1, "e1_r%d.front", file_num );
} /* end if */
if ( ( fp2 = fopen( file1, "w" ) ) == NULL ) {
    fprintf( stderr, "Error opening file %s.", file1 );
    exit(1);
} /* end if */
for ( i=0; i<(total*count); i++ ) { /* write data to the
*.front file */
    fprintf(fp2, "%3.14f ", *(answer_data+i));
    if ( (i+1)%total==0 ) { /*would mean finished a whole row*/
        fprintf(fp2, "\n");
    } /* end if */
} /* end for */
fclose(fp2);

free(answer_data); /*free memory*/
printf( "%s non-dominated set cardinality = %d\n", file1, count
);
} /* end is_clone */

```


Appendix F: Raw Data and Experimental Statistics

Table 9. Raw Data for Final Generational Distance

Replication	G								
	BB Size 2	BB Size 4	BB Size 8	Init Pop Size 1200	Init Pop Size 600	Pr Cut .2	Pr Cut 0	Pr Splice .6	Pr Splice .8
1	125.115451	512.8011	297.8076	364.311	567.4849	98.31657	568.9251	651.2109	196.6754
2	125.991964	142.7817	532.89	873.2793	376.0693	114.1282	327.9645	589.7849	580.2024
3	193.653321	503.6903	764.5851	122.3463	142.7817	155.1454	154.2427	374.2282	165.0465
4	633.966632	175.569	883.9351	219.5416	453.9992	369.8408	382.3122	330.4212	96.85462
5	117.822819	584.0015	90.74446	302.5113	342.3077	460.1379	200.7084	784.5899	130.2442
6	438.490448	96.24376	91.05122	831.0963	647.7359	164.3747	122.7495	775.1324	506.5935
7	388.436987	206.8894	233.6241	778.8981	341.0263	471.8852	221.3943	80.39486	202.3626
8	156.39923	126.5027	666.4384	165.4059	120.591	88.11754	92.89454	201.8199	288.6717
9	594.410855	282.3055	457.6109	262.3954	34.45147	1263.971	180.8423	282.465	173.7748
10	53.4061256	401.6491	76.84958	226.7083	273.2529	359.7219	92.53599	55.83325	260.0223
11	402.541745	541.5588	428.9971	308.7174	376.478	223.016	90.94767	94.99117	78.83749
12	639.554108	137.4583	272.3274	201.6548	1152.456	55.83325	98.58962	256.6123	157.2979
13	536.547361	183.9366	554.1251	327.3591	471.688	466.4903	351.3493	235.2073	764.138
14	518.327715	399.7562	104.4964	83.53441	74.61097	512.924	401.8493	366.0302	88.41957
15	425.893687	55.83325	449.1533	509.8431	98.58142	262.6975	341.4665	755.1439	85.95738
16	588.075281	98.85652	336.5296	313.8191	156.8742	273.3299	148.8832	1072.582	714.2348
17	106.27138	106.0025	194.1108	258.6733	130.3585	164.7798	92.91047	687.6016	115.3837
18	1297.55844	125.3886	1311.962	10.21327	282.1049	93.32948	508.5393	1138.673	243.5371
19	101.905519	131.722	111.2674	107.8312	92.59555	484.5023	317.0348	493.2018	45.52391
20	80.0994277	961.8244	311.7048	166.6124	95.85525	105.0631	77.57315	107.2334	254.968
21	599.61571	273.5505	375.6198	158.6287	264.8213	107.0121	98.48881	302.7405	87.53758
22	159.856698	1452.21	203.25	474.8541	114.8377	162.8288	196.9644	474.7975	197.6486
23	73.755233	269.6512	321.6703	221.3695	535.3442	1590.665	122.9531	296.5017	739.7947
24	503.92885	302.3389	483.0555	206.9495	403.7624	631.3033	350.1023	388.8342	142.6813
25	410.20264	1322.523	28.37985	365.8437	82.01277	350.4857	127.8845	88.62832	85.97397
26	79.579773	225.325	546.5883	107.4325	30.67919	1289.051	87.43287	306.5166	127.8894
27	209.024357	204.8656	435.7566	497.7318	113.8987	121.0588	55.83325	461.6742	557.8949
28	76.4774978	688.3423	70.5623	75.26208	582.8499	410.3384	446.1906	424.3851	625.766
29	253.884067	817.7795	253.2699	216.4271	174.3479	120.9857	104.9711	156.9337	895.1915
30	759.942846	376.0639	239.2597	78.53964	96.06281	642.4688	357.0865	359.8707	144.9781

Table 10. Descriptive Statistics for Final Generational Distance

G	BB Size 2	BB Size 4	BB Size 8	Init Pop Size 1200	Init Pop Size 600
Mean	355.0245389	390.2473495	370.9207517	294.5930059	287.6640022
Standard Error	51.46720685	64.42525301	50.70024185	40.12471071	44.59797423
Median	321.160527	271.6008249	316.6875801	224.0388789	219.5846086
Mode	#N/A	#N/A	#N/A	#N/A	#N/A
Standard Deviation	281.8975016	352.8716435	277.6966613	219.7720917	244.273165
Sample Variance	79466.20142	124518.3968	77115.4357	48299.7723	59669.37916
Kurtosis	2.686647286	2.79371578	3.38349507	1.616715948	4.021731942
Skewness	1.317141708	1.745867391	1.522534257	1.411791894	1.685114119
Range	1244.152316	1396.376753	1283.582077	863.065988	1121.77701
Minimum	53.40612564	55.83325178	28.37984578	10.21326632	30.67918813
Maximum	1297.558441	1452.210005	1311.961924	873.2792544	1152.456198
Sum	10650.73617	11707.42049	11127.62255	8837.790178	8629.920065
Count	30	30	30	30	30
Largest(1)	1297.558441	1452.210005	1311.961924	873.2792544	1152.456198
Smallest(1)	53.40612564	55.83325178	28.37984578	10.21326632	30.67918813
Confidence Level(95.0%)	105.2623145	131.7645091	103.6936941	82.06429251	91.21314864

G	Pr Cut 2	Pr Cut 0	Pr Splice .6	Pr Splice .8
Mean	387.1267494	224.0540115	419.8013029	291.8033947
Standard Error	69.51649012	26.60529804	51.36087249	45.44044169
Median	268.013722	167.5425163	362.9504463	185.2251064
Mode	#N/A	#N/A	#N/A	#N/A
Standard Deviation	380.7574976	145.7232189	281.3150843	248.8875494
Sample Variance	144976.272	21235.25652	79138.17668	61945.01224
Kurtosis	3.591832146	-0.53083871	0.572682719	-0.026245448
Skewness	1.953085113	0.781647157	0.965171287	1.161375263
Range	1534.831446	513.0918206	1082.839354	849.6675571
Minimum	55.83325178	55.83325178	55.83325178	45.52391375
Maximum	1590.664698	568.9250723	1138.672605	895.1914708
Sum	11613.80248	6721.620345	12594.03909	8754.10184
Count	30	30	30	30
Largest(1)	1590.664698	568.9250723	1138.672605	895.1914708
Smallest(1)	55.83325178	55.83325178	55.83325178	45.52391375
Confidence Level(95.0%)	142.1772638	54.41397388	105.0448362	92.93618901

Table 11. Raw Data for Overall Nondominated Vector Generation

ONVG									
Replication	BB Size 2	BB Size 4	BB Size 8	Init Pop Size 1200	Init Pop Size 600	Pr Cut 2	Pr Cut 0	Pr Splice .6	Pr Splice .8
1	5	5	4	4	6	7	6	5	6
2	4	3	4	2	3	3	5	3	6
3	3	8	4	4	3	5	11	5	1
4	7	7	5	5	7	7	6	3	6
5	2	3	4	1	7	5	4	6	4
6	4	3	4	6	7	3	3	5	6
7	7	5	6	5	8	9	8	2	4
8	3	3	5	5	5	6	4	7	6
9	4	6	4	3	3	4	3	8	4
10	4	5	3	5	12	5	2	1	5
11	5	7	5	3	4	4	3	6	6
12	3	3	6	4	3	1	6	6	3
13	6	3	5	3	11	6	11	7	7
14	5	7	4	4	5	7	6	5	7
15	6	1	5	4	6	9	7	4	5
16	3	5	4	4	5	5	8	2	6
17	5	5	5	6	4	3	5	7	3
18	6	5	6	1	4	6	5	4	7
19	4	3	3	7	5	5	6	5	5
20	6	4	4	5	6	5	5	4	4
21	6	7	2	3	4	6	5	4	2
22	9	7	6	5	4	9	8	6	7
23	5	7	5	8	6	3	5	6	7
24	4	6	2	7	6	8	5	6	3
25	7	5	2	4	6	5	3	3	6
26	3	6	3	4	3	8	5	4	3
27	5	6	6	5	4	3	1	6	6
28	6	5	3	3	3	6	4	6	7
29	4	3	5	3	5	4	5	2	5
30	5	8	7	6	3	4	3	6	7

Table 12. Descriptive Statistics for Overall Nondominated Vector Generation

ONVG	BB Size 2	BB Size 4	BB Size 8	Init Pop Size 1200	Init Pop Size 600
Mean	4.866666667	5.033333333	4.366666667	4.3	5.266666667
Standard Error	0.28257166	0.330389298	0.237241491	0.3	0.406649708
Median	5	5	4	4	5
Mode	5	5	4	4	3
Standard Deviation	1.547708725	1.809616712	1.29942516	1.643167673	2.227312178
Sample Variance	2.395402299	3.274712644	1.688505747	2.7	4.96091954
Kurtosis	0.330279614	-0.76336707	-0.46288338	0.170352801	2.559001346
Skewness	0.47692497	-0.20260917	-0.1404731	0.08094422	1.465651956
Range	7	7	5	7	9
Minimum	2	1	2	1	3
Maximum	9	8	7	8	12
Sum	146	151	131	129	158
Count	30	30	30	30	30
Largest(1)	9	8	7	8	12
Smallest(1)	2	1	2	1	3
Confidence Level(95.0%)	0.577924251	0.675722354	0.485213594	0.613559227	0.83169249

ONVG	Pr Cut 2	Pr Cut 0	Pr Splice 6	Pr Splice 8
Mean	5.366666667	5.266666667	4.8	5.133333333
Standard Error	0.369788176	0.420545167	0.31948234	0.306006285
Median	5	5	5	6
Mode	5	5	6	6
Standard Deviation	2.025413254	2.303420745	1.749876843	1.676065453
Sample Variance	4.102298851	5.305747126	3.062068966	2.809195402
Kurtosis	-0.37390166	1.068685333	-0.5311948	-0.285311721
Skewness	0.154107933	0.811198657	-0.41701776	-0.742760169
Range	8	10	7	6
Minimum	1	1	1	1
Maximum	9	11	8	7
Sum	161	158	144	154
Count	30	30	30	30
Largest(1)	9	11	8	7
Smallest(1)	1	1	1	1
Confidence Level(95.0%)	0.756302151	0.860111911	0.653415108	0.625853467

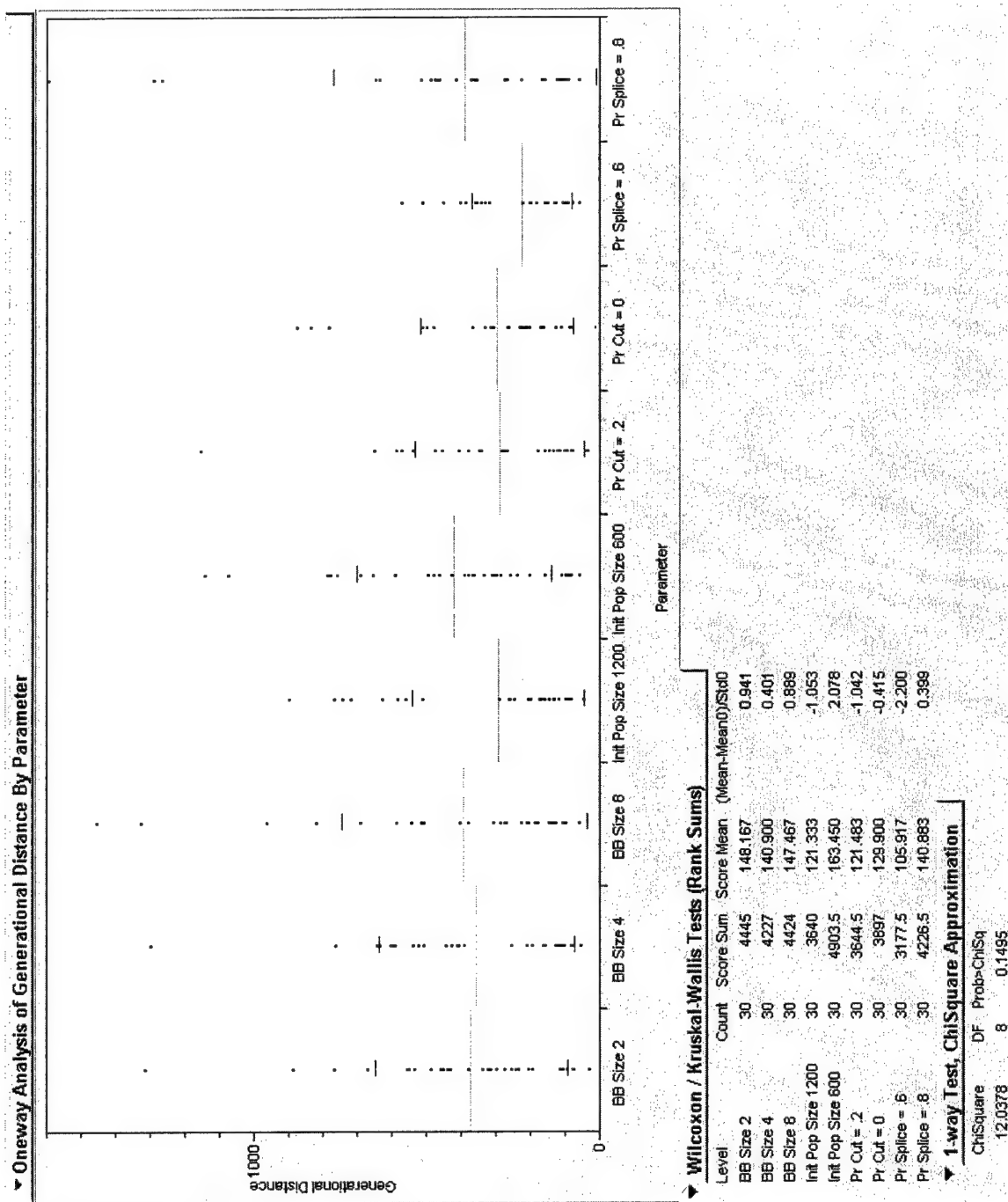


Figure 13. Kruskal-Wallis H -Test Results for Final Generational Distance

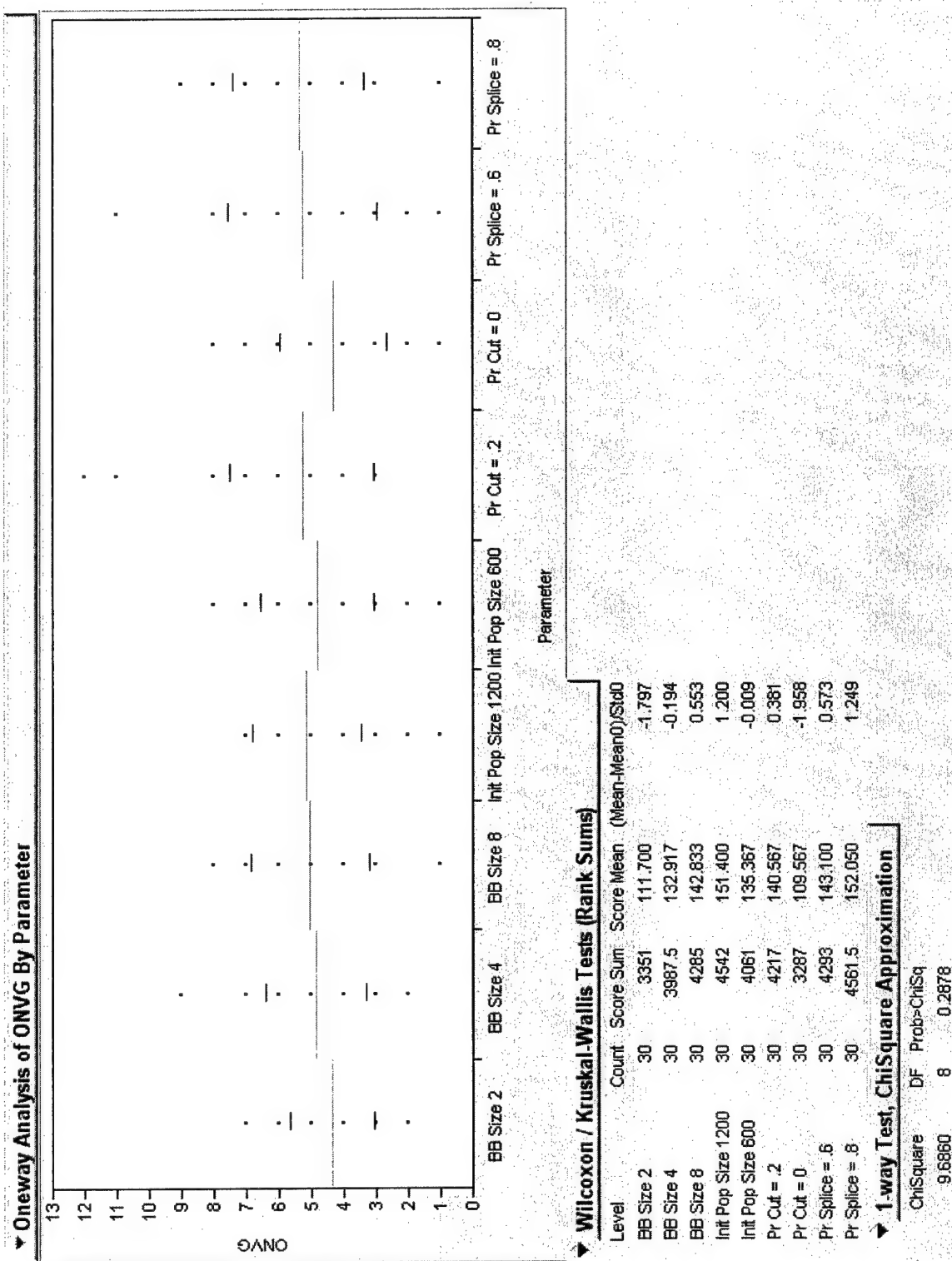


Figure 14. Kruskal-Wallis H -Test Results for Overall Nondominated Vector Generation

Appendix G: 3D Plots of PF_{true} and PF_{known} Using Alternative Parameter Values

The following figures plot in three dimensions the PF_{true} with the PF_{known} generated by the implicitly constraining MOMGA-II using the alternative parameter values specified in the experimental design section of Chapter III. The plots are intended to show that without employing any explicit constraint handling methods, the MOMGA-II output approximates the structure of PF_{true} .

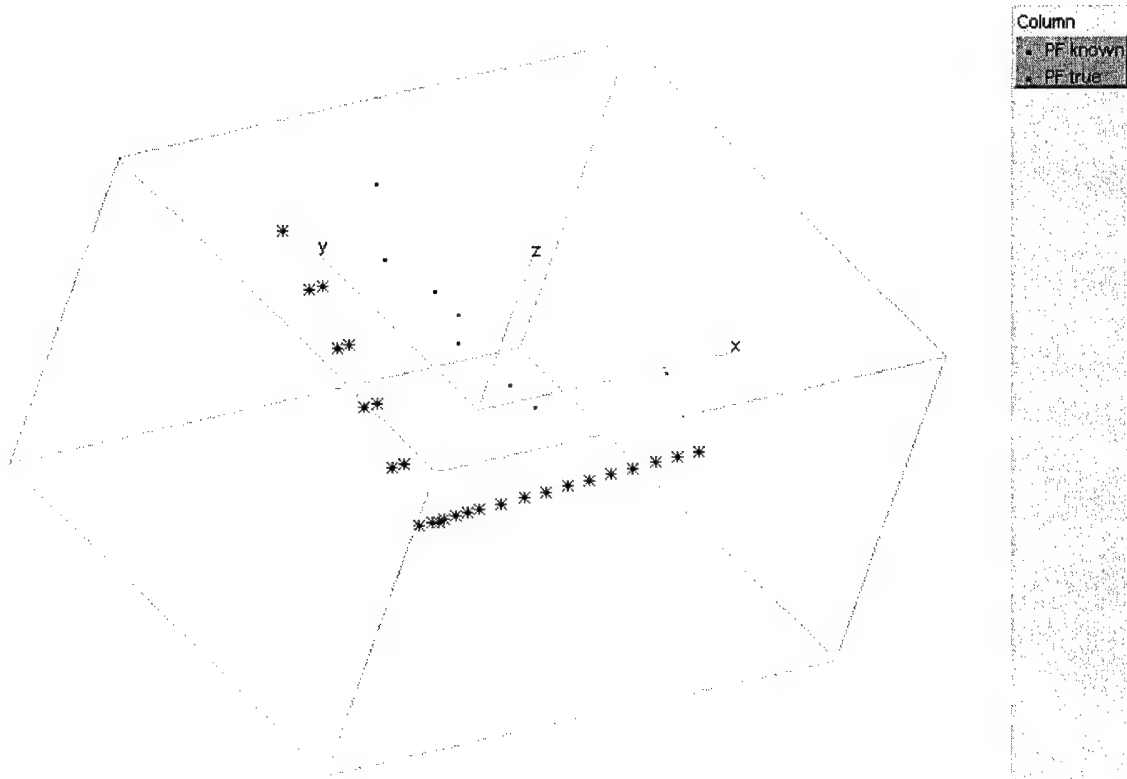


Figure 15. Plot of PF_{true} and PF_{known} for BB size 4

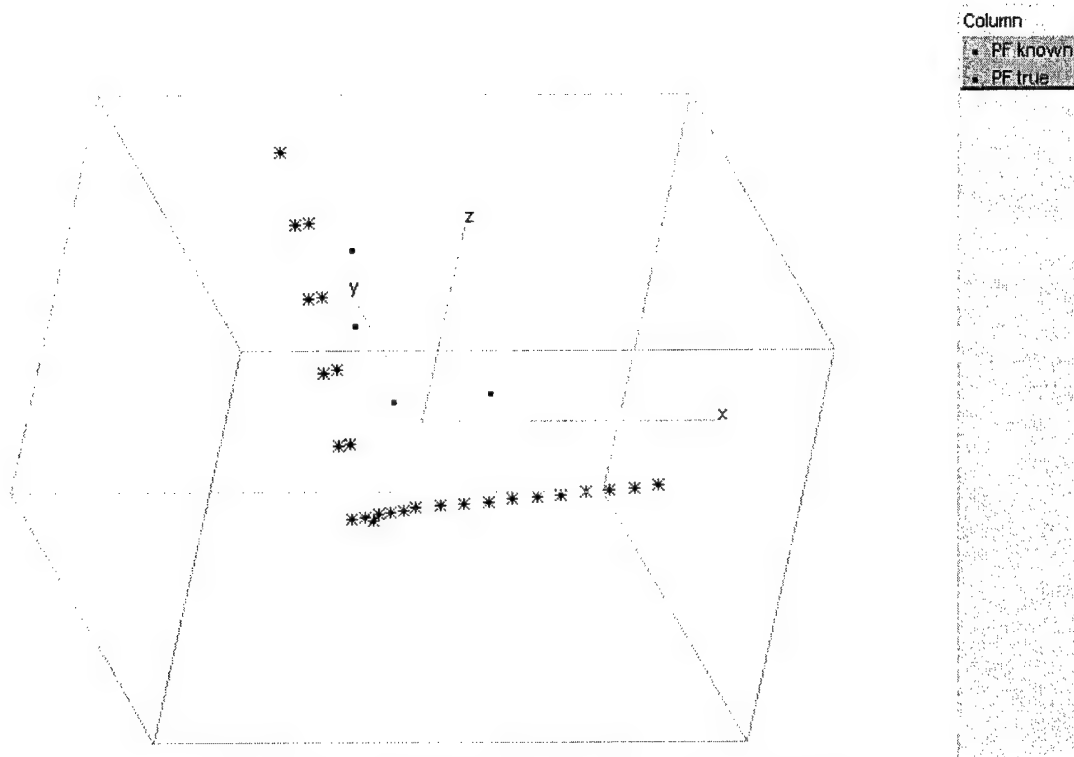


Figure 16. Plot of PF_{true} and PF_{known} for BB size 8

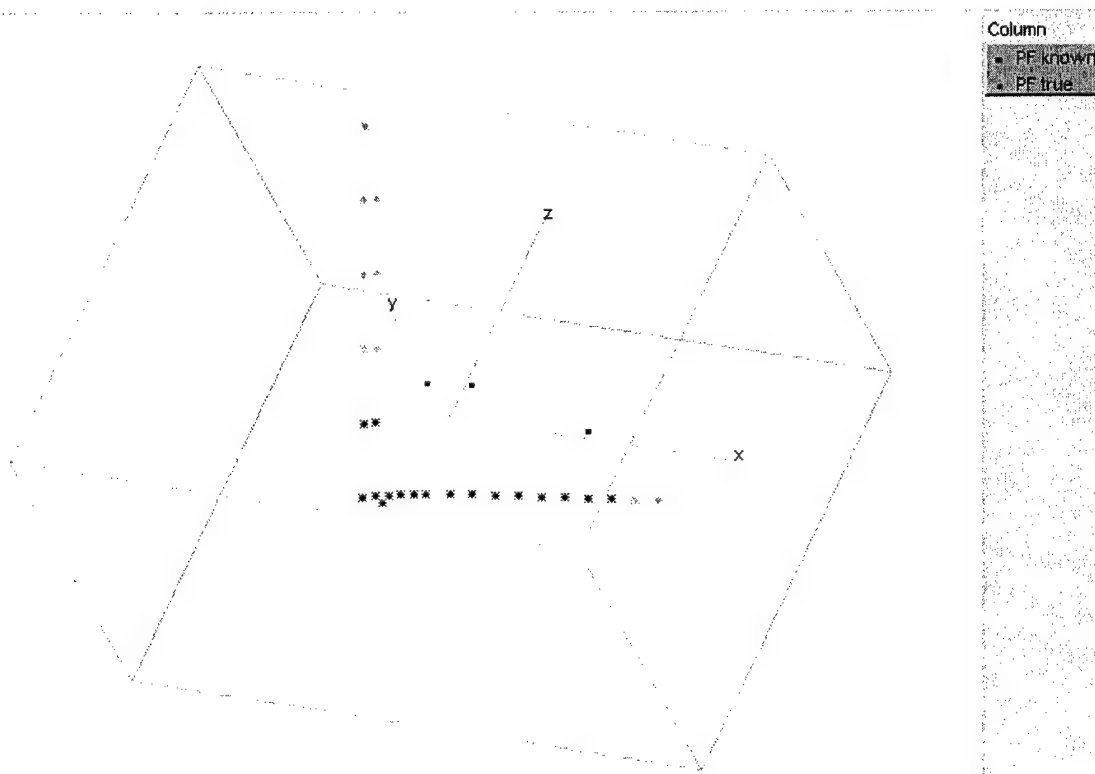


Figure 17. Plot of PF_{true} and PF_{known} for BB size 2

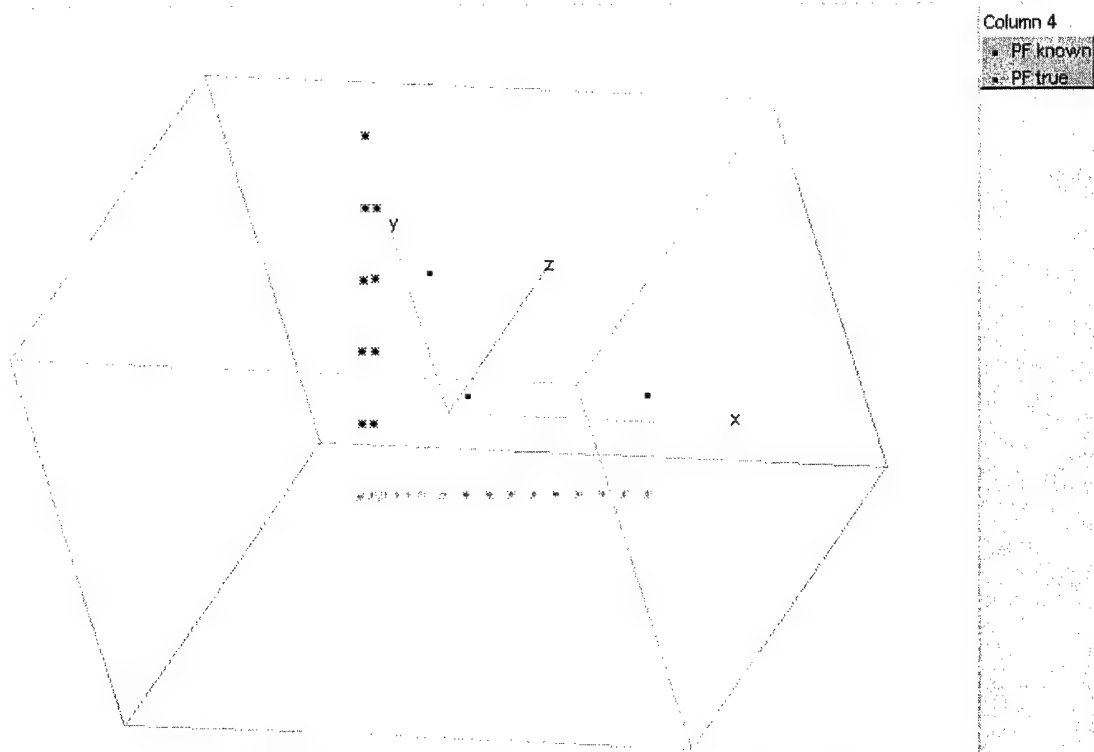


Figure 18. Plot of PF_{true} and PF_{known} for $P_{cut} = 0$

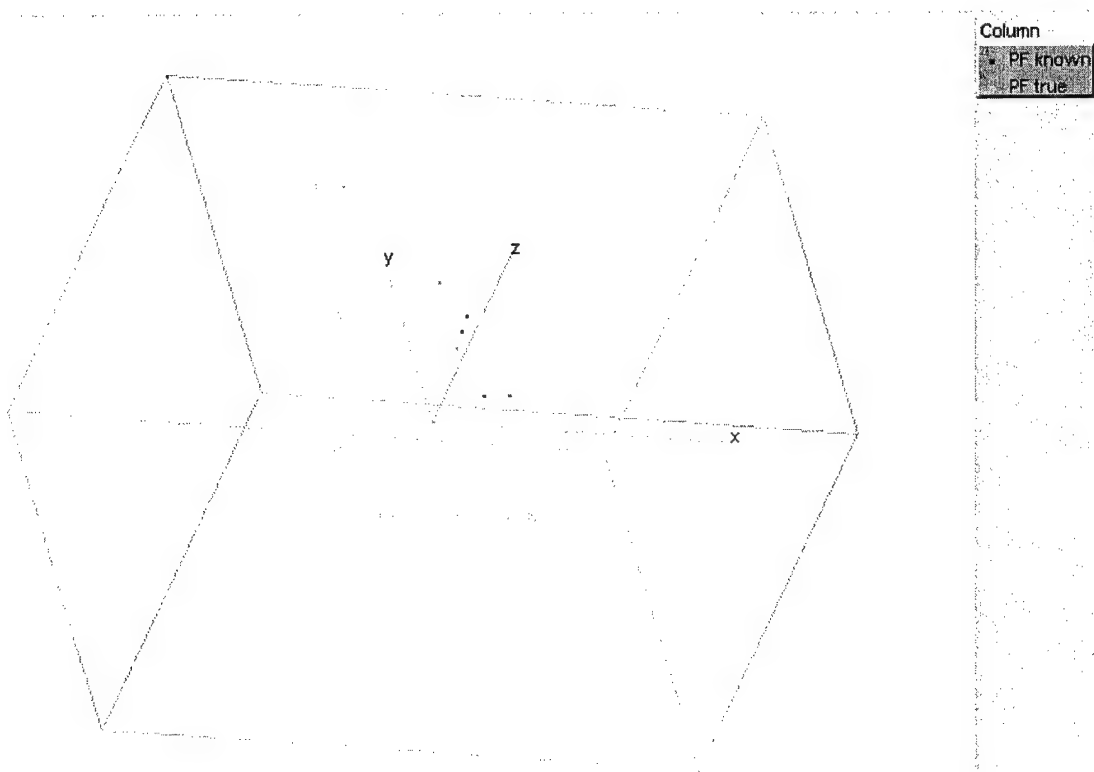


Figure 19. Plot of PF_{true} and PF_{known} for $P_{cut} = 0.2$

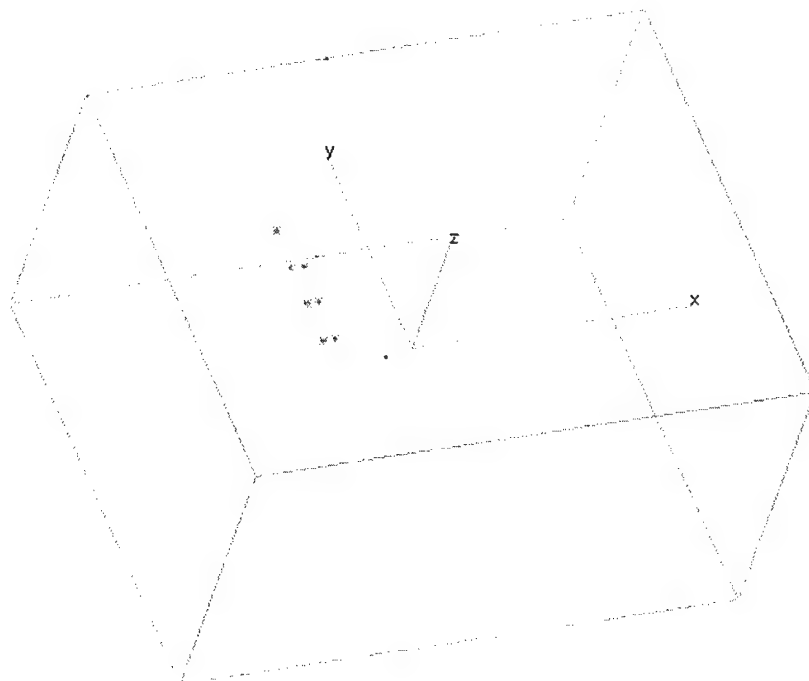


Figure 20. Plot of PF_{true} and PF_{known} for $P_{splice} = 0.8$

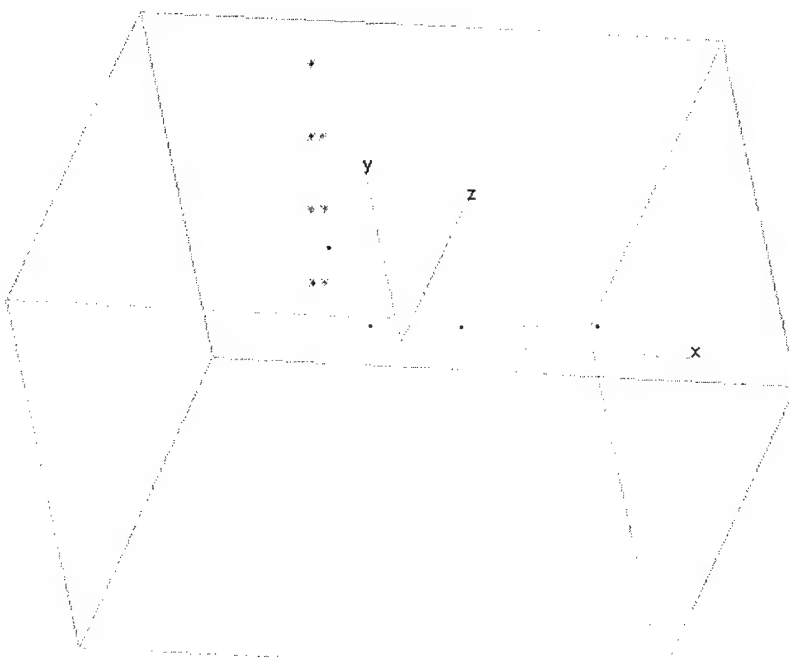


Figure 21. Plot of PF_{true} and PF_{known} for $P_{splice} = 0.6$

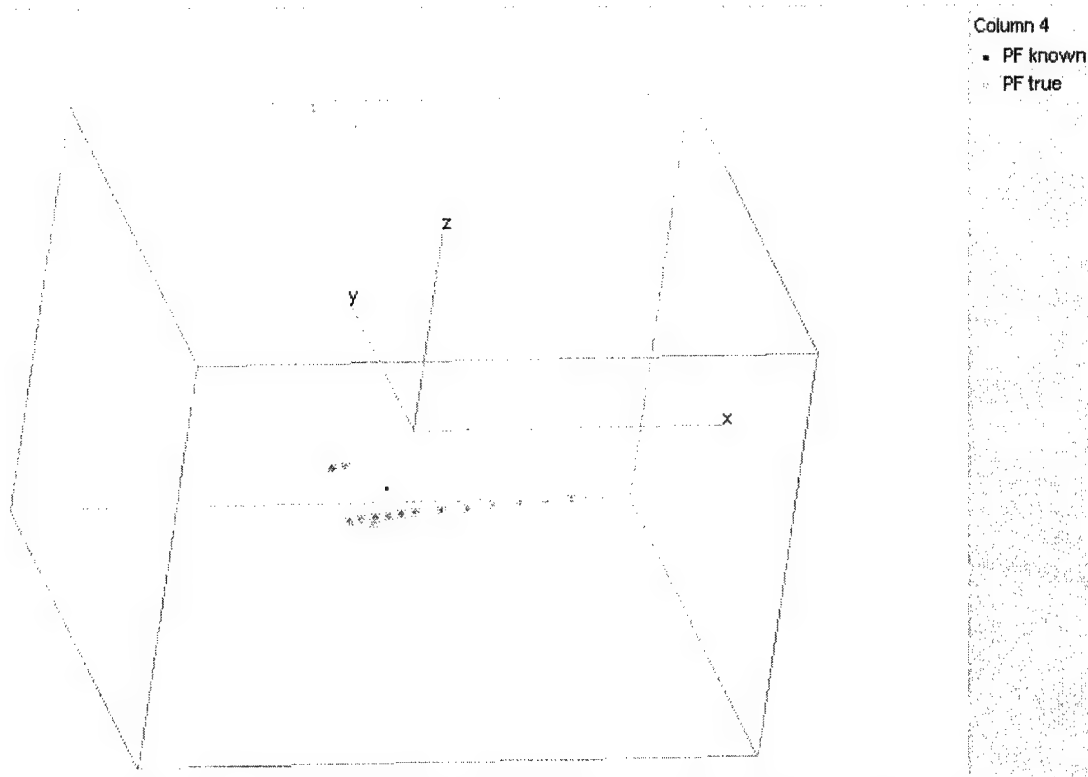


Figure 22. Plot of PF_{true} and PF_{known} for initial population size = 600

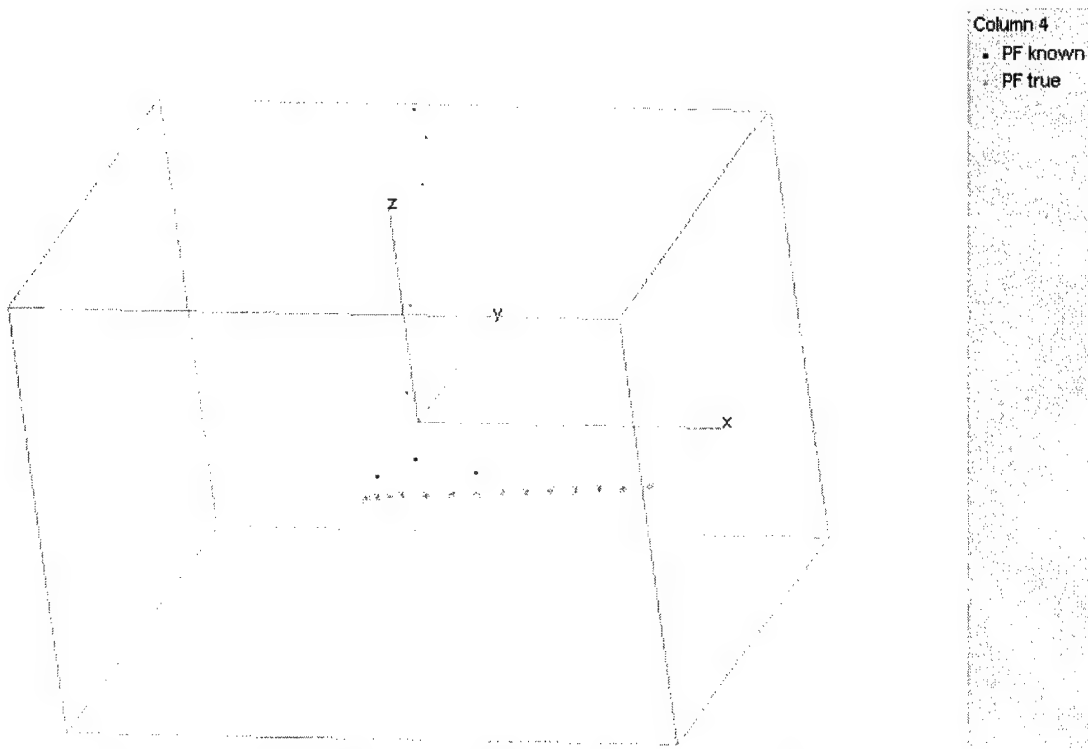


Figure 23. Plot of PF_{true} and PF_{known} for initial population size = 1200

Appendix H: 3D Plots of PF_{known} for Resource Levels 1–5

The following figures plot in three dimensions for each Resource Level the PF_{known} generated by the explicitly constraining MOMGA-II using the basic parameter values specified in the experimental design section of Chapter III.

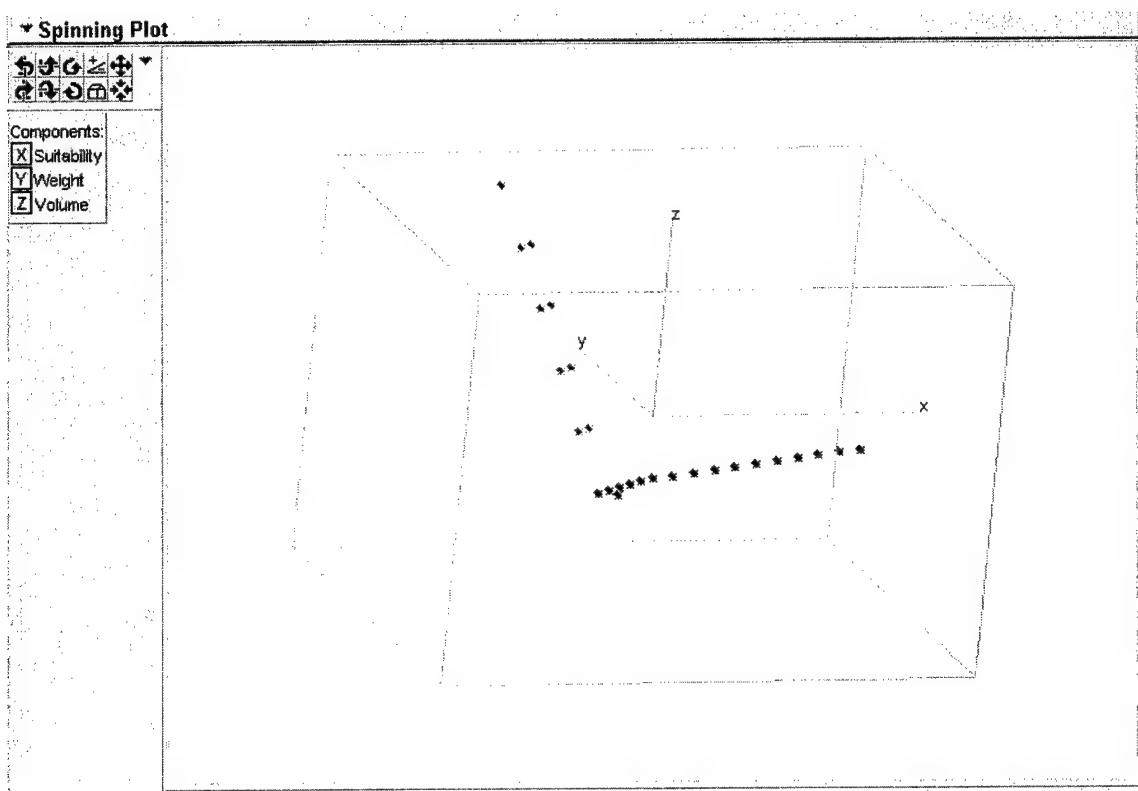


Figure 24. PF_{known} Plotted With PF_{true} for Resource Level 1

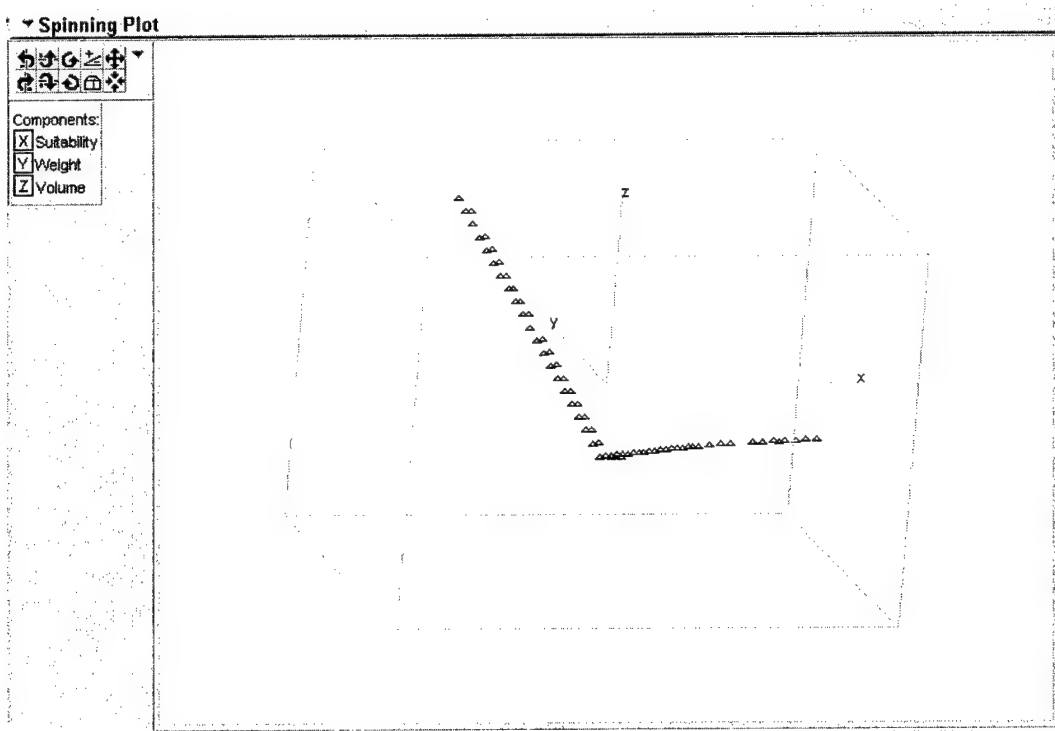


Figure 25. PF_{known} Plotted for Resource Level 2

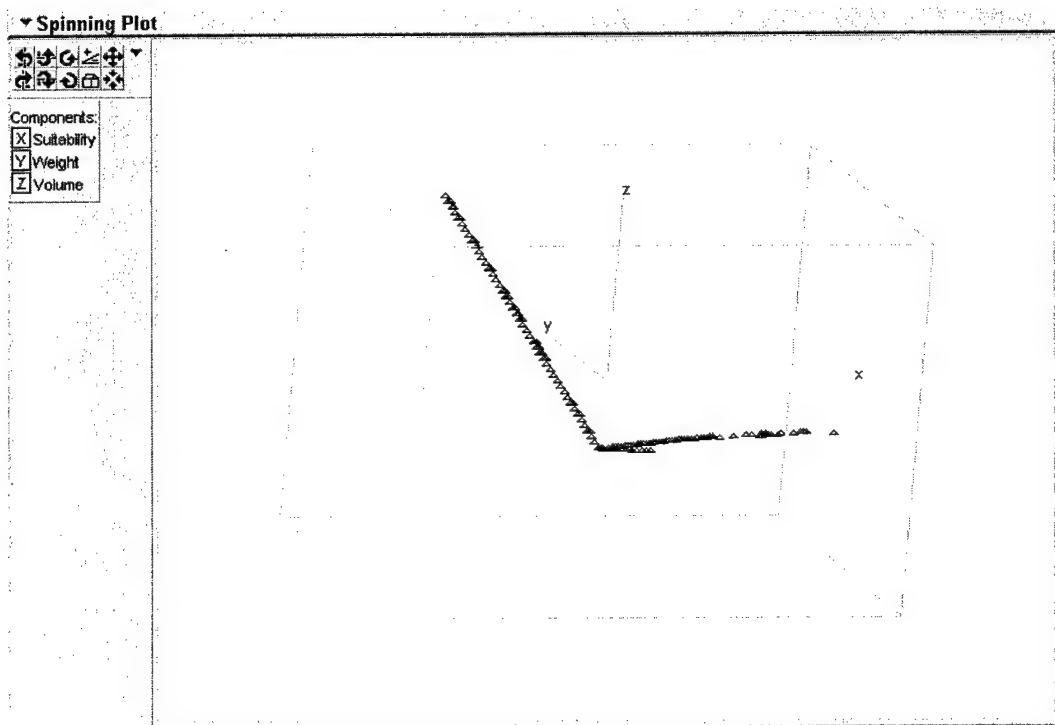


Figure 26. PF_{known} Plotted for Resource Level 3

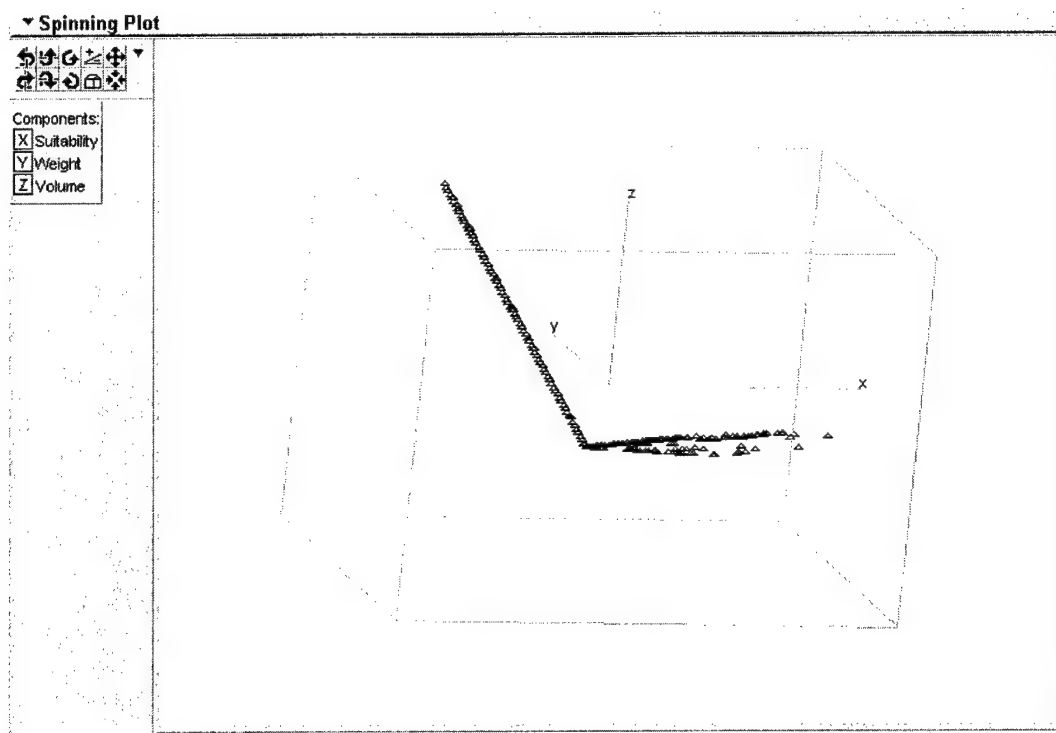


Figure 27. PF_{known} Plotted for Resource Level 4

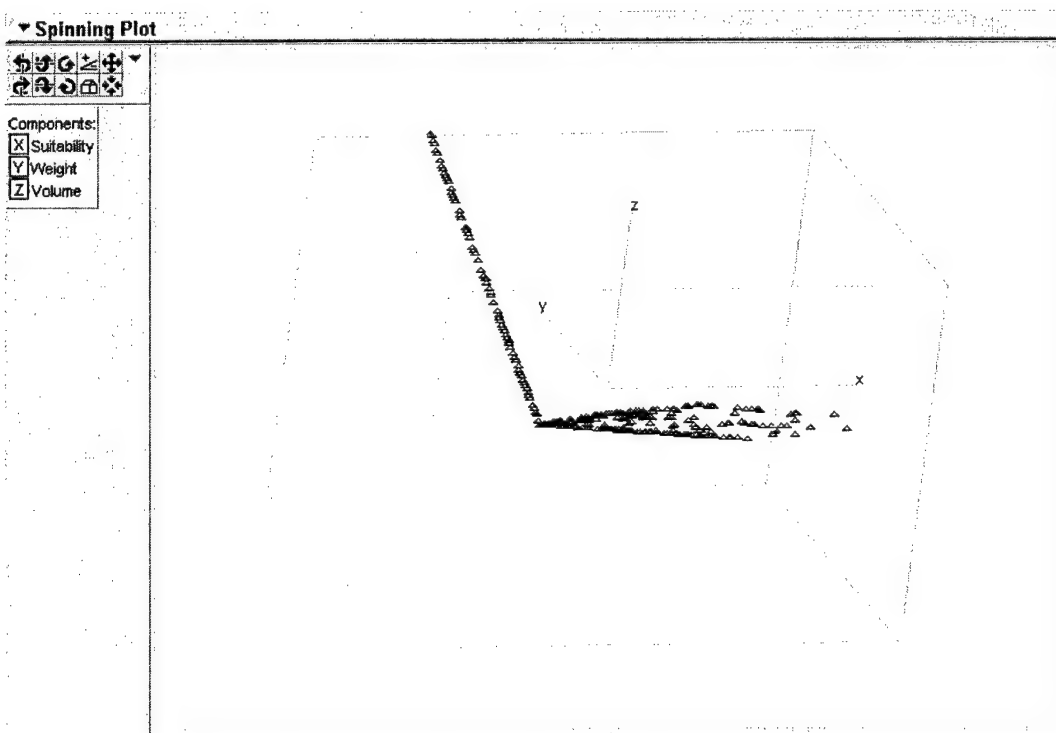


Figure 28. PF_{known} Plotted for Resource Level 5

Bibliography

- Azar, Shapour, Brian J. Reynolds, and Sanjay Narayanan. "Comparison of Two Multiobjective Optimization Techniques With and Within Genetic Algorithms," Proceeding of the 1999 ASME Design Engineering Technical Conferences. Las Vegas NV, 12-15 September 1999.
- Baumol, W.J. and R.C. Bushnell. "Error Produced by Linearization in Mathematical Programming," Econometrica, 35: 447-471 (1967).
- Bäck, Thomas. Evolutionary Algorithms in Theory and Practice. New York: Oxford University Press, 1996.
- Bentley, P.J. and J.P. Wakefield. Finding Acceptable Solutions in the Pareto Optimal Range using Multiobjective Genetic Algorithms. Dept. of Computer Science, Univ. College London, U.K., 1999.
- Buzo, Christopher D. A Decision Support Tool to Aid Campaign Planners in Selecting Combat Aircraft for Theater Crisis. MS Thesis, AFIT/GEE/ENS/00M-02. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2000.
- Carrico, Todd M. "ALP Overview: Background." Advanced Logistics Project. United States, Defense Advanced Research Projects Agency. 6 Apr. 2000. n. pag. 10 June 2000. http://www.darpa.mil/iso/alp/Public_Access/Overview/index.htm
- Caryl, Matthew. Mutants. no date. n. pag. 20 December 2000. <http://www.catachan.demon.co.uk/Projects/MUTANTS>
- Chipperfield, Andrew and Peter Fleming. Evolutionary Computation Research: An Overview of Evolutionary Algorithms for Control Systems Engineering. Dept. of Automatic Control and Systems Engineering, University of Sheffield, England. 18 April 2000. n. pag. 24 November 2000. <http://www.shef.ac.uk/~gaipp/control1.html>
- Coello Coello, Carlos A. "A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques," Knowledge and Information Systems, 1: 269-308 (August 1999).
- Darwin, Charles. The Origin of Species by Means of Natural Selection; or, The Preservation of Favored Races in the Struggle for Life and The Descent of Man and Selection in Relation to Sex. First Modern Library edition. New York, 1936.

- Deb, Kalyanmoy Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. Dept of Mechanical Engineering, Indian Institute of Technology Kanpur, India, 1998.
- Deb, Kalyanmoy An Introduction to Genetic Algorithms. Dept of Mechanical Engineering, Indian Institute of Technology Kanpur, India, 1999.
- Deb, Kalyanmoy Multi-Objective Evolutionary Algorithms: Introducing Bias Among Pareto-Optimal Solutions. Dept of Mechanical Engineering, Indian Institute of Technology Kanpur, India, 1999.
- Deb, Kalyanmoy and Samir Agrawal. Understanding Interactions Among Genetic Algorithm Parameters. Kanpur Genetic Algorithms Laboratory, Indian Institute of Technology, Kanpur, India, 1999.
- Deb, Kalyanmoy, Samir Agrawal, Amrit Pratap, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multiobjective Optimization: NSGA-II. Kanpur Genetic Algorithms Laboratory, Indian Institute of Technology, Kanpur, India, 2000.
- Dorigo, Marco and Vittorio Maniezzo. "Parallel Genetic Algorithms: Introduction and Overview of Current Research," Parallel Genetic Algorithms: Theory and Applications. Ed. Joachim Stender. Amsterdam, Netherlands: IOS Press, 1993. (ISSN: 0922-6389).
- Ehrgott, Matthias and Xavier Gandibleux. An Annotated Bibliography of Multiobjective Combinatorial Optimization. Fachbereich Mathematik, Universitat Kaiserslautern, Germany, 13 April 2000.
- Goldberg, David E., Kalyanmoy Deb, Hillol Kargupta, and Georges Harik. Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms. Illinois Genetic Algorithms Laboratory, Dept. of General Engineering, Univ. of Illinois at Urbana-Champaign, Urbana IL, February 1993.
- Goldberg, David E., Kalyanmoy Deb, and Bradley Korb. "Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale," Complex Systems, 4:415-444 (1990).
- Goldberg, David E., Bradley Korb, and Kalyanmoy Deb. "Messy Genetic Algorithms: Motivation, Analysis, and First Results," Complex Systems, 3:493-530 (1989).
- Goddard, Matthew W. Estimating Deployed Airlift and Equipment Requirements for F-16 Aircraft in Support of the Advanced Logistics Project. MS Thesis, AFIT/GLM/ENS/01M-11. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2001.

- Gray Perry, William Hart, Laura Painton, Cindy Phillips, Mike Trahan, and John Wagner. "Evolutionary Algorithms – General Information," A Survey of Global Optimization Methods. Sandia National Laboratories, Albuquerque NM. 10 March 1997. n. pag. November 24, 2000.
<http://www.cs.sandia.gov/opt/survey/ts.html>.
- Filcek, Paul G. A Quantitative Decision Support Model to Aid Selection of Combat Aircraft Force Mixes for Contingency Deployment. MS Thesis, AFIT/GLM/ENS/01M-10. School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 2001.
- Fonseca, Carlos M. and Peter J. Fleming. "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization," Genetic Algorithms: Proceeding of the Fifth International Conference. ed. S. Forrest. San Mateo, CA: Morgan Kaufmann, July 1993.
- Fonseca, Carlos M. and Peter J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization. Dept. of Automatic Control and Systems Engineering, Univ. of Sheffield, U.K., 19 May 1995.
- Harel, David. Algorithmics: The Spirit of Computing. Cornwall, Eng: TJ Press, 1987.
- Hoffman, Frank. Messy Genetic Algorithms. 14 May 1997. n. pag. <http://www.ang-physik.uni-kiel.de/~hoeft/mga.html>.
- Horn, Jeffrey, Nicholas Nafpliotis, and David E. Goldberg. Multiobjective Optimization Using the Niche Pareto Genetic Algorithm. Dept of General Engineering, University of Illinois at Urbana-Champaign, July 1993.
- Johnson, Alan W. Assistant Professor of Logistics Management, Dept. of Operational Sciences, Air Force Institute of Tech, Wright-Patterson AFB OH. Personal interview. 13 February 2001.
- Johnson, Alan W. and Stephen M. Swartz. AFIT Research IN support of DARPA's Advanced Logistics Project. Unpublished briefing. April 5, 2000a.
- Johnson, Alan W. and Stephen M. Swartz. Mission-Resource Value Assessment Technique. Unpublished briefing. ALP Winter Conference. dec00.L&R.ppt December 2000b.
- Judge, Paul J. The Potential Influence of Advanced Logistics on Defense Air Transportation. Air Force Inst Of Tech, Wright-Patterson AFB OH, School Of Logistics And Acquisition Management, June 1998. (AD-A354260).
- Kargupta, Hillol. Messy Genetic Algorithms: Foundations, Future Directions, and Applications. n. pag. Computational Methods Group, Los Alamos National Laboratory, Los Alamos NM, no date.

- Knowles, Joshua. and David Corne. "The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Multiobjective Optimisation,' Proceedings of the 1999 Congress on Evolutionary Computation. Piscataway, NJ: IEEE Service Center, 98-105, 1999.
- Knuth, D.E., The Art of Computer Programming, 2nd Edition, Vol. 2, Addison-Wesley, 1981. (ISBN 0-201-03822-6)
- Lynn, Larry, "DARPA's Advanced Logistics Program," National Conference on Setting an Intermodal Transportation Research Framework. Conference Proceeding 12:13-22. (4-5 March 1996). Washington: National Academy Press, 1997.
- Matthews, James K. and Cora J. Holt. So Many, So Much, So Far, So Fast. Washington: GPO, 1996.
- Mathematical Optimization. no date. n. pag. 6 October 2000.
<http://www.ccs.uky.edu/csep/MO/NODE2.html>.
- McClave, James T., P. George Benson, and Terry Sinchich. Statistics for Business and Economics (7th Edition). Upper Saddle River NJ: Prentice Hall, 1998.
- Muczyk, Jan P. "The Changing Nature of External Threats, Economic and Political Imperatives, and Seamless Logistics," Airpower Journal: 81-92 (Summer 1997).
- Osyczka A. and S. Kundu. "A New Method to Solve Generalized Multicriteria Optimization Problems Using the Simple Genetic Algorithm," Structural Optimization, 10: 94-99 (1995).
- Practical Guide to Genetic Algorithms. Chemometrics Research Group, Naval Research Laboratory, Washington D.C. no date. n. pag. 20 December 2000.
<http://chemdiv-www.nrl.navy.mil/6110/sensors/chemometrics/practga.html>
- Rappe, Andrew M. and Eric J. Walter. The Maxwell-Boltzmann Distribution: A Hands on Approach. Dept. of Chemistry, Univ. of Pennsylvania, Philadelphia PA. no date. n. pag. 5 March 2000.
<http://oobleck.chem.upenn.edu/~rappe/MB/MBmain.html>
- Reeves, Colin R. Modern Heuristic Techniques for Combinatorial Problems. London, Eng: McGraw-Hill, 1995.
- Reynolds, Dan E. Assistant Professor of Statistics, Dept. of Mathematics and Statistics, Air Force Institute of Tech, Wright-Patterson AFB OH. Personal interview. 25 February 2001.

- Ruiz-Andio, Alvaro, Lourdes Arauho, Fernando Sáenz, and José Ruz. "A Hybrid Evolutionary Approach for Solving Constrained Optimization Problems over Finite Domains," IEEE Transactions on Evolutionary Computation, 4: 353-372 (November 2000).
- Sawaragi, Yoshikazu, Hirotaka Nakayama, and Tetsuzo Tanino. Theory of Multiobjective Optimization. Orlando FL: Academic Press Inc, 1985.
- Shaw, Jane. Evolutionary Computation Research. 2 November 1998. n. pag. 24 November 2000. <http://www.shed.ac.uk/~gaipp/mogas.html>
- Srinivas, N. and Kalyanmoy Deb. "Multi-Objective Function Optimization Using Non-Dominated Sorting Genetic Algorithms," Evolutionary Computation, 2(3):221-248 (1995).
- Swartz, Stephen. "ALP Pilot Problem and Derivation of Mathematical Model." Unpublished report. Wright-Patterson AFB OH, 1999.
- Taber, John T; Balling, Richard; Brown, Michael R; Day, Kirsten; Meyer, Gregory A. "Optimizing Transportation Infrastructure Planning with a Multiobjective Genetic Algorithm Model." Transportation research record. n. 1685. (1999) pp. 51. (ISSN: 0361-1981)
- Van Veldhuizen, David A. Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations. PhD Thesis, AFIT/DS/ENG/99-01, Air Force Institute of Technology, Wright-Patterson AFB OH.
- Van Veldhuizen, David A. and Gary B. Lamont. "Genetic Algorithms, Building Blocks, and Multiobjective Optimization," Genetic and Evolutionary Computation Conference, July 1999.
- Van Veldhuizen, David A. and Gary B. Lamont. "Multiobjective Evolutionary Algorithms: Analyzing the State-of-the-Art," Evolutionary Computation, 8(2): 125-147 (2000).
- Van Veldhuizen, David A. and Gary B. Lamont. "On Measuring Multiobjective Evolutionary Algorithm Performance," IEEE 2000 Congress on Evolutionary Computation, 16 – 19 July 2000.
- Williams, Thomas J. The Canvas and the Clock - Impact of Logistics at the Operational Level of War. Naval War College, Newport RI, Dept. of Operations, 17 May 1993. (AD-A266897).
- Wolpert, David H. and William G. Macready. No Free Lunch Theorems for Search. The Santa Fe Institute, Santa Fe NM. 23 February 1996.
- Yu, Po-Lung. Multiple-Criteria Decision Making. New York: Plenum Press, 1985.

- Zitzler, Eckart, Kalyanmoy Deb, and Lothar Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results (Revised Version). Computer Engineering and Networks Laboratory, Dept. of Electrical Engineering, Swiss Federal Institute of Technology, Zurich, Switzerland. 22 December 1999.
- Zitzler, Eckart and Lothar Thiele. Multiobjective Optimization Using Evolutionary Algorithms—A Comparative Case Study. Swiss Federal Institute of Technology, Zurich, Switzerland, 1998.
- Zydallis, Jesse B. PhD student, Dept. of Computer Engineering, Air Force Institute of Tech, Wright-Patterson AFB OH. Personal interview. 8 February 2001.
- Zydallis, Jesse B., David A. Van Veldhuizen, and Gary B. Lamont. “A Statistical Comparison of Multiobjective Evolutionary Algorithms Including the MOMGA-II,” 1st International Conference on Multi-Criterion Optimization, Zurich, Switzerland, 7-9 March 2001.

Vita

On 19 April 1984, then Airman Wakefield enlisted in the U.S. Air Force and entered training as an Avionics Navigation Systems Specialist. From 1985 to 1992, he was stationed at Norton AFB, California where he worked on a wide variety of aircraft including the T-39 and C-141B. In September 1992, he transferred to March AFB, California where he worked as a Communication and Navigation Systems Specialist on the KC-10A. While on active duty, he earned a Bachelor of Science degree in Mathematics from California State University, San Bernardino in 1993.

In November 1994, he entered Officer Training School. Upon commissioning, he served as an aircraft maintenance officer at Davis-Monthan AFB, Arizona. During his two-and-a-half year stay, he was a flight commander for the 358th Fighter Squadron and the 355th Component Repair Squadron. In January 1998, Capt Wakefield was assigned as base plans officer for the 24th Wing at Howard AFB, Panama. His tour there culminated with the successful handover of the installation to the Panamanian government in accordance with the Carter-Torrijos Treaty of 1977.

In August 1999, he entered the Graduate Logistics Management program at the Air Force Institute of Technology. Upon graduation, he will be assigned to the ICBM System Program Office at Hill AFB, Utah

REPORT DOCUMENTATION PAGE				<i>Form Approved</i> OMB No. 074-0188					
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.									
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.									
1. REPORT DATE (DD-MM-YYYY) 20-03-2001		2. REPORT TYPE Master's Thesis		3. DATES COVERED (From - To) 1 Mar 2000 - 20 Mar 2001					
4. TITLE AND SUBTITLE IDENTIFICATION OF PREFERRED OPERATIONAL PLAN FORCE MIXES USING A MULTIOBJECTIVE METHODOLOGY TO OPTIMIZE RESOURCE SUITABILITY AND LIFT COST				5a. CONTRACT NUMBER					
				5b. GRANT NUMBER					
				5c. PROGRAM ELEMENT NUMBER					
6. AUTHOR(S) Wakefield, David J. Jr., Capt, USAF				5d. PROJECT NUMBER					
				5e. TASK NUMBER					
				5f. WORK UNIT NUMBER					
7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Building 640 WPAFB OH 45433-8865				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GLM/ENS/01M-24					
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Dr. Todd Carrico DARPA/ISO 3701 North Fairfax Drive Arlington, Virginia 22203-1714 (703) 526-6616				10. SPONSOR/MONITOR'S ACRONYM(S)					
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)					
12. DISTRIBUTION/AVAILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.									
13. SUPPLEMENTARY NOTES									
14. ABSTRACT AFIT research in support of the Advanced Logistics Project is directed at developing a Mission-Resource Value Assessment Tool for rationally assigning relative value to resources and identifying alternative force mixes to logistics and operational planners. Research of factors that affect force mix composition has been strictly limited to how the operating environment of USAF combat aircraft influences their performance in specified aerospace missions. In contrast, this research makes use of an aircraft's designed suitability to perform specified aerospace missions in order to examine the tradeoff between mission suitability and the amount of lift needed to deploy and operate the asset. An Evolutionary approach was applied to a tri-objective constrained optimization problem with 15 decision variables with the goal of producing five Pareto optimal sets of force mixes corresponding to five progressively larger sortie capability levels. Analysis of the results include absolute performance comparisons using different operating parameter settings, and time complexity in relation to problem scale. Preliminary results were also generated from a version of the algorithm that uses a solution repair function. These results help to assess the viability of using a multi-objective fast messy genetic algorithm to identify well balanced force mixes.									
15. SUBJECT TERMS Optimization, Military Planning, Defense Planning, Air Force Planning, Logistics Planning, Logistics Management, Heuristic Methods									
16. SECURITY CLASSIFICATION OF: <table border="1" style="width: 100%; border-collapse: collapse; margin-top: 5px;"> <tr> <td style="width: 33%; padding: 2px;">a. REPORT U</td> <td style="width: 33%; padding: 2px;">b. ABSTRACT U</td> <td style="width: 33%; padding: 2px;">c. THIS PAGE U</td> </tr> </table>			a. REPORT U	b. ABSTRACT U	c. THIS PAGE U	17. LIMITATION OF ABSTRACT <div style="text-align: center; font-weight: bold; font-size: 1.2em;">UU</div>		18. NUMBER OF PAGES <div style="text-align: center; font-weight: bold; font-size: 1.2em;">138</div>	
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U							
			19a. NAME OF RESPONSIBLE PERSON Lt Col Alan W. Johnson 19b. TELEPHONE NUMBER (Include area code) (937) 255-6565, ext 4284 Alan.Johnson@afit.af.edu						